ual: creative computing institute

Individual Project

University of the Arts London

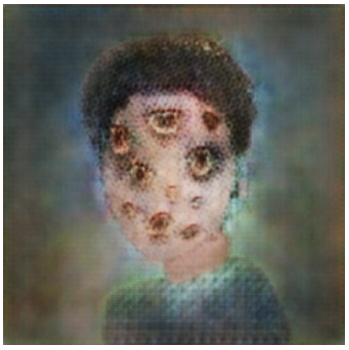
Creative Computing Institute

Developing an Interactive Machine Learning Tool for Artists

Author:
Hans Smith-Wrinch

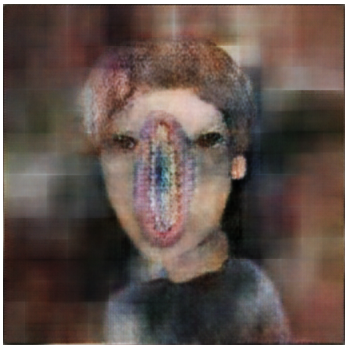
Supervisor:
Prof. Mick Grierson

November 19, 2021



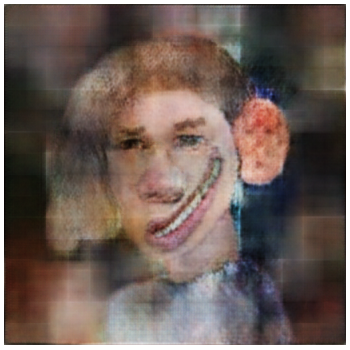
Abstract

This work sets out to develop an end-to-end interactive machine learning tool for artists in an attempt to democratise and educate on machine learning. A program is proposed supporting dataset and architecture configuration, training (with live training feedback), and various interactive capabilities. Novel modes of inference are put forward, including new forms of latent vector representation and manipulation - and first of its kind GAN to image to image translation and image to image to image to image translation. Two user studies are conducted on a prototype of the software, detailing a survey and a conversation with an artist. The work culminates in seven new pieces of art created with the software.



Acknowledgements

I would like to thank my little brother Nick, for letting me use his computer. A huge thanks also goes to my supervisor Mick, for always giving me a meeting to look forward to.



Contents

Chapter 1

Introduction

In recent times, there has been an explosion of growth within the Artificial Intelligence (AI) and Machine Learning (ML) research sector. The introduction of radical technology often inspires change in the art world. One can draw comparisons to the invention of the camera; provoking artists to hijack this technology to spawn a radically new and popular art-form within photography[35]. It comes as no surprise that an increasing amount of creatives are using new ML-based generative techniques to innovate and produce exciting works of art. This includes the creation of music[23], representations of dance[45], automatic image stylisation[30], poetry[78], and rapid image to image interpretation[38] to produce realistic and bespoke landscapes[64], portraits[52], and cats[1]. Many of these works provide the unique opportunity to uncover abstractions engrained in modern technologies. For example, artist Anna Ridler utilised Generative Adversarial Networks (GANs) in her work *Mosaic Virus*[69] to explore the economic speculations surrounding cryptocurrency (see Figure 1.1). Ridler complimented this project by showcasing the dataset she used in *Myriad (Tulips)*[67]. This featured thousands of hand labelled photographs of Tulips that were used to train her GAN, in an effort to remind the viewer that data is physical, handpicked, and representative. By making the dataset an artwork and unveiling ML's invisible backbone, she exaggerates the necessity of social responsibility required within this technology. Unfortunately, related oversights in responsibility continue to enforce racial[20] and sexual[12] discrimination.

Figure 1.1: *Mosiac Virus. Anna Ridler, 2019* - As part of a three part series: *Myriad, Mosiac Virus,* and *Bloemenveiling*[68]. Within *Bloemenveiling* the tulips were sold on the ethereum blockchain. Inspired by the Netherland's tulip mania in the 1630s, where the price of tulips soared to the price of a house before dramatically crashing, in parallel to the nature of the cryptocurrency market. Over the span of a week the generated artefacts would dynamically change using interpolations on the input latent vector (see Section X). At the end of the week they would die/disappear. "In this way, the decay and impermanence of the natural world are reintroduced into the digital world"[2]

In another work, Joy Buolamwini directly challenged racial and sexual discrimination as a result of algorithmic bias whilst developing *Aspire Mirror*[13]. This object used AI-enabled facial analysis technology to change the reflection of an onlooker. She noticed it had a hard time following her face consistently. Her later *UpBeat Walls*[14] project had the same issue, significantly favouring the faces of lighter-skinned individuals to her own. It was only after she wore a white face mask that her software responded correctly. These observations culminated in her landmark thesis *Gender Shades* (see Figure 1.2), revealing clear bias in datasets extensively used in the mainstream[15]. Projects like these, coupled with the current boom of Non-Fungible Tokens (NFTs) being employed in the digital art marketplace[31], mean computer generated art - specifically ML-based art - continues to capture public attention. A large proportion of the public, however, remain unaware of ML as a process and (crucially) the ethical implications of big data[44]. In a study conducted by IPSOS MORI[37], just 9% of the 978 participants were familiar with the term *machine learning*. In the same study, 75% who recognised an application of machine learning said this was from mainstream media, compared to 21% from entertainment (and even smaller still from art). There is no doubt, with so many people being informed through mainstream media, there is the potential to garner fearmongering surrounding ML/AI[18]. Having explicit public education in the topic is paramount[11] towards solving these issues. The difficulty of this education necessitates a multi-faceted approach to tackle the public discourse in ML/AI[62].

Figure 1.2: Installation photo from the Barbican's AI: More than Human exhibition featuring Joy Buolomwini's Gender Shades project, Barbican Centre, 2019

By empowering more artists to represent and unravel ML abstractions in their work, we could provide a unique opportunity to educate more people on the processes involved in ML, AI, and other modern technologies[27]. This facilitates a motivation to afford creative practitioners the capability to generate art and innovate using ML techniques. In most cases, however, extensive knowledge of programming and ML has been required by creatives to utilise ML in their works. Detrimentally, formal education in these subjects is not accessible to many artists. Interactive Machine Learning (IML) has since been hugely successful in enabling many artists to produce works with ML[55]. As a result, this work aims to lower the barrier of entry for artists to produce ML-based art through an IML approach. By giving artists IML software with a focus on usability, this work hopes to offer an extensive tool-set to educate and be educated by ML.

1.1 Objectives

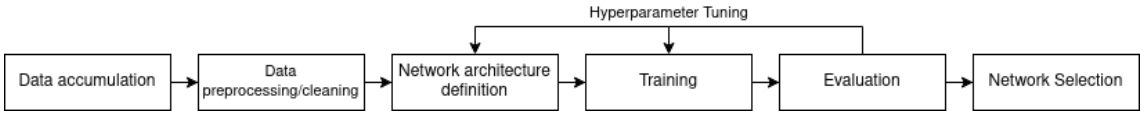
The primary goal of this research is to produce an end-to-end tool to create ML generated art with an emphasis on usability for non-technical artists. This software should provide: i.) minimum dependencies ii.) easy installation iii.) live feedback on training iv.) interactive output v.) custom inputs vi.) good user centred design vii.) easy export.

Chapter 2

Background

2.1 ML techniques

We can identify 7 main steps in the process of developing a conventional ML algorithm:



1. Data accumulation

ML models require data related to the problem we are trying to solve. For instance, let's say we were employing a CNN that predicts a breed of cat from on an image. We would collect images of cats coupled with a *label* describing the breed. Instead, if we were developing a LSTM[] network to generate Haikus, we would accumulate textual datapoints of Haiku poems. When all the observations in the dataset are unlabelled, such as in the Haiku generation problem, it is *unsupervised learning*. Inversely, when the data is paired with a label like the cat breed prediction problem, it is *supervised learning*. Sometimes this can involve tens or hundreds of thousands of datapoints, particularly in applications featuring deep networks. To accommodate this, there exists open datasets such as ImageNet[21], featuring over a million labelled images. Often, public datasets fail to cater to a project's needs. This is usually the case when a model relies on niche or protected data. A manual approach is necessary through web scraping or a more labour intensive method[67]. Consequently, this can prove to be one of the most time costly stages.

2. Data preprocessing/cleaning

The data must then be compiled into a dataset that is usable for the network. Typically, the dataset is randomly partitioned into a training and testing set. The considerably smaller testing set is used to evaluate network performance and is, fundamentally, never *seen* when training. In some cases unwanted artefacts may remain in the dataset. This can include datapoints with missing or false elements. To prevent noise from negatively affecting model performance, these must be filtered out in what is known as data cleaning. The cleaning method is highly dependant on the format in which the noise presents itself. The dataset may also need to be prepared for the networks input specification or for increased training efficacy. For example, resizing images to a networks input dimensions or normalizing the data. This can happen at run-time or as a preliminary action.

3. Network architecture definition

Ml algorithms implement neural network(s) which facilitate a complex mapping between the given input data and a desired output format. Conventional neural networks consist of a series of layers: the input layer, hidden layer(s), and the output layer (see Figure 2.1). The hidden layers contain neurons with weight attributes that are learned through training. These neurons may be fully connected or partially connected to the adjacent layers. The output layer may contain one or

more neurons. These neurons may represent unbounded values, such as in a regression problem, or predictions, such as in a classification problem. These values are typically calculated from using activation function. Activation functions constitute the bridge between the previous layer's values and the desired output. Selecting the correct activation is critical[34]. Modern neural networks typically use non-linear activation functions. Common non-linear activation functions include: *sigmoid* - used in binary classification; $softmax$ - used in non-binary classification; *relu* - used in networks with many layers. In practice, more sophisticated networks are used instead of the conventional neural network. These networks usually follow an abstraction of this process. To illustrate, CNNs feature more complex layer types between the input and output layers. One type is the Convolutional layer which is specifically designed to extract features in high dimensional data such as images. The learned weights in this layer are found in the matrix elements of the kernel (see Section 2.1.2). Common layer types include: Pooling layers - used to downsample feature maps; Dropout layers - used to reduce overfitting; Normalization Layers - used to normalize neuron outputs in deep networks. Deciding on the correct configuration for a network is largely based on experimentation, intuition, and inspiration from other works.

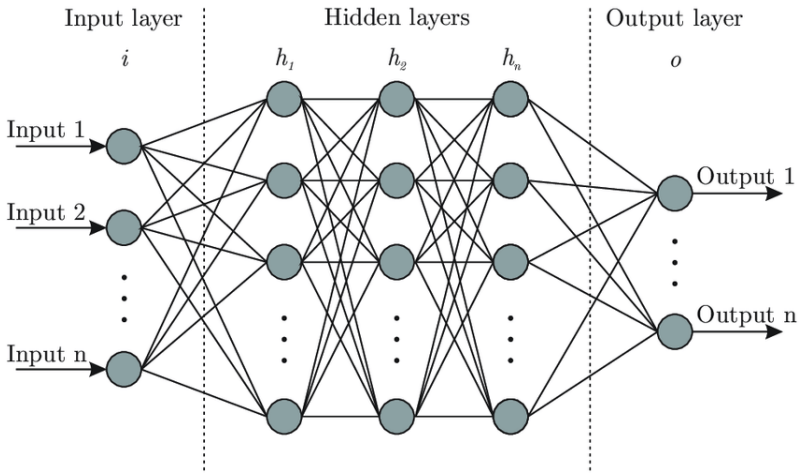


Figure 2.1: Conventional Neural Network architecture

4. Training

Network training involves adjusting weight values to yield an optimal output. It is an iterative operation comprised of two main stages: forward propagation and backward propagation. The former refers to the calculation of intermediate variables throughout the network from input layer to output layer. The evaluated output is compared to the target output (or the ground truth). A measure of difference between these values is calculated using a loss function. Selection of loss function can be highly influential in model performance[63]. Common loss functions include: L1 or L2 loss - for regression; hinge or cross entropy loss - for classification. Within more sophisticated systems, a bespoke loss function may be necessary[33]. The second stage refers to the adjustment of weights in a pass from output layer to input layer - this is where the learning takes place. The backpropogation algorithm functions to perturb the weights so as to minimize the calculated loss. This is accomplished through an optimisation method such as stochastic gradient descent[70]. Other optimisers include ADAM[47] and Adagrad[24]. Similar to the loss function, the choice of the optimiser is an important aspect in determining training efficacy. Each forward and backward pass constitutes an *epoch* and is repeated until an arbitrary amount of epochs has passed.

5. Evaluation

After training, the model must be evaluated to gauge its performance with out of sample data. Only considering the losses at each training step, whilst providing some insight, is insufficient. Even with low training losses, the model may have modelled the training data too well through memorisation - referred to as *overfitting* (see Figure 2.2). This means noise in the training data is learned; negatively impacting the models ability to generalise to new data. To properly evaluate the model, the testing set (unseen until now) is inputted into the network. Various evaluation

metrics are then employed to quantify the models performance. For regression, L2 loss is a popular metric. Classification metrics are less straightforward. A simple metric is accuracy (ie the ratio of correct predictions to number of predictions). Confusion matrices may provide a more illustrative description of performance. Another popular metric is F1 score, which considers the precision and recall of prediction results.

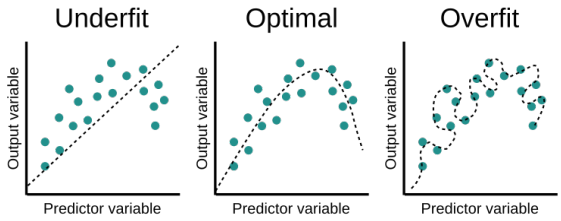


Figure 2.2: An example of underfitting and overfitting. Underfitting can be spotted without an unseen test set as it will usually be picked up as a poor loss value.

6. Hyperparameter tuning

In ML, a hyperparameter is a variable that is used to control training. This variable can affect the speed and quality of the learning process and is configured external to the model before training. For example, a common hyperparameter is *batch size*, which determines the amount of data points sent through the model at each training epoch. *Learning rate* is another hyperparameter which affects the extent that the weights are adjusted during training. Hyperparameters include variables which affect the network structure (eg. the number of hidden layers). Setting hyperparameters correctly is important since they can directly impact a model's performance[19]. This is a difficult task, however, as they cannot usually be inferred from the data and architecture alone. By repeatedly training the model with different hyperparameter configurations, the optimal values can be chosen based on the best evaluation metrics. It is worth noting, sometimes the dataset has an extra partition, the *validation set*, which is used to tune the hyperparameters.

7. Model selection

After evaluating many individual models, the best model is selected based on the required criteria. There may be numerous factors which can affect the final chosen model. These include complexity, maintainability, and available resources. Generally however, model performance is the main concern.

2.1.1 Transfer Learning

Transfer learning is a technique that enables models to benefit from the knowledge of another model(s). This normally means building a new model on top of a previously trained model. By using a pretrained model as a starting point, we can simplify the ML problem. This may present itself as reducing the training data required and/or reducing training time.

2.1.2 Convolutional Neural Networks

CNNs[50] specialise in extracting features from high dimensional data, such as images. This is due to their convolutional layers (see Figure 2.1.2). Convolutions use a kernel to extract certain features from an input image. A kernel is a small matrix that contains values which weight the output of each pixel with respect to its adjacent pixels. The features this extracts can result in effects like a sharpened image, blurred image, and more valuably, an image's edges[76]. Each convolution is ordinarily followed by a pooling layer to downsample intermediary outputs which affords more efficient computation. As a result, CNNs are commonly used in image classification tasks[49].

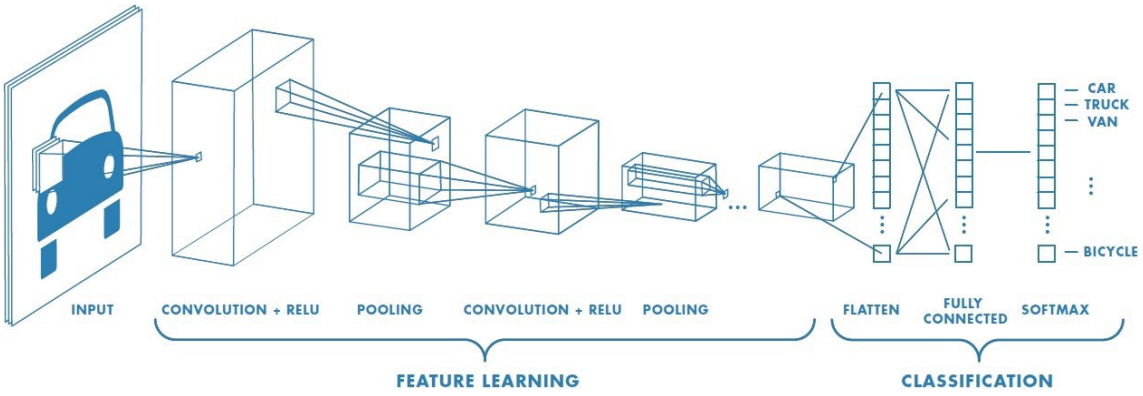


Figure 2.3: Convolutional Neural Network architecture.

2.1.3 Generative Methods

Style Transfer

Style transfer generates an image with the content from one image in the style from another. This has the effect of "painting" an image in a given style (see Figure 2.4). It works by taking advantage of a CNNs hierarchical feature representation: lower level features are extracted closer to the input layers and higher level features closer to the output. Lower layers capture capture rudimentary artefacts such as horizontal and vertical lines. Higher layers capture more complex objects such as faces or dogs. Thus, lower levels encapsulate the style of an image and high levels, the content of an image. Style transfer is accomplished by inputting noise into a pretrained CNN; the original paper[30] uses VGG-19[75]. The noise image is optimised using a cost function constituting style and content loss. Style loss involves lower layers in the network and the style image. Content loss involves higher layers in the network and the content image. We say the noise image is optimised as the pixels change within it, whilst the weights of the model stay the same.

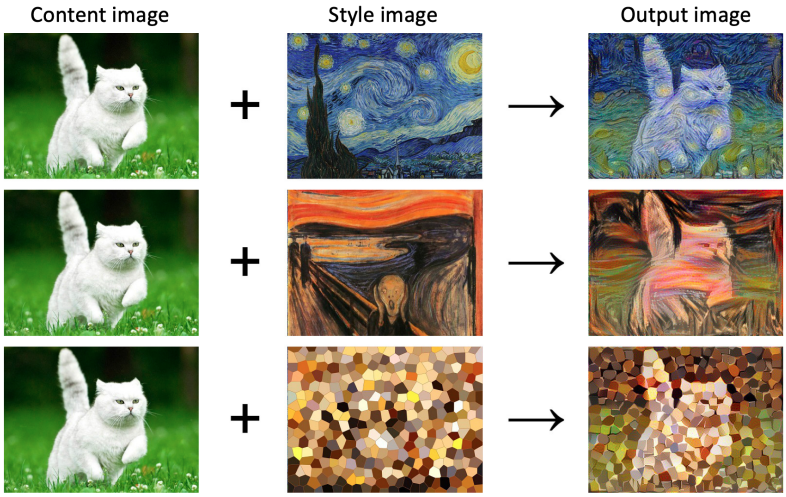


Figure 2.4: Example of Style Transfer

Autoencoders

Autoencoders may be used to generate images. They are an unsupervised learning algorithm that transform high dimensional data into a low dimensional latent vector space (encoder) before reconstructing the input data from this representation (decoder). Put succinctly, an image is encoded and subsequently decoded through a bottleneck architecture (see Figure 2.5). After training, images (or other high dimensional data) may be generated using the decoder by removing the encoder. Images may then be generated by inputting a random latent vector into the decoder.

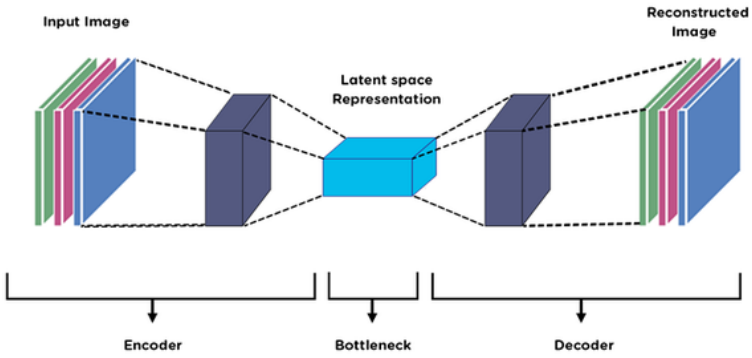


Figure 2.5: Autoencoder architecture

$$\min_{D} \max_{G} (E_{x \sim p_{data}(x)}[\log (D(x))] + E_{z \sim p_z(z)}[\log (1 - D(G(z)))])$$

Figure 2.6: GAN cost function represented as a minimax/zero-sum game

Interpolations on this vector can result in continuous perturbations or "movement" on the output image (see Section 2.1.3). It is worth mentioning that Variational Autoencoders, which contain a more sophisticated bottleneck, generally facilitate more desirable interpolation outputs[48].

Generative Adversarial Networks

GANs[33] are among the most popular generative methods. They perform better than autoencoders for complex tasks, but they require more data to train from and can be notoriously difficult to train (to convergence)[51]. A GAN is an unsupervised learning algorithm that cleverly frames the problem as supervised learning through the use of two sub-models: the discriminator and the generator. GANs typically work with image data and use CNNs for these sub-models (often referred to as DCGANs[66]). The discriminator reads an image and predicts whether it is part of the original dataset. The generator reads a latent vector and generates an image that, when inputted into the discriminator, maximises the evaluated probability of it being from the original dataset. The discriminator and generator are trained in two distinct steps. In the first step, the generator generates a batch of samples and these, along with a batch of real samples (from the dataset), are classified by the discriminator to be real or fake. The discriminator's weights are then updated to optimise correct predictions in subsequent epochs. In the second step, the generator's weights are updated based on how well it managed to fool the discriminator (ie based on the cost function). Using game theory, the GANs training process can be identified as a two player zero-sum game between the discriminator and the generator. This is reflected in the original paper's[33] cost function in Figure 2.6. We can analogise the GAN to the discriminator being the police and the generator being an art forger. The art forger wants to trick the police. The police want to improve their forged art detection skills to catch out the forger. When the art forgerer gets caught by the police, it uses this feedback to improve their methods. It is a zero sum game because as one agent improves, the other agent worsens by an equivalent amount. There has since been improvements on this schema[5]. Variants of the GAN include: conditional GAN[59] (CGAN) - which outputs according to a parameter; Progressive GAN[42] - which specialises in outputting large images; and Pix2PixGAN[38].

Pix2Pix

Image to image translation is the complex transformation of a source image to a target image, with one image often in a distinct coded representation. To illustrate, these transformations may constitute satellite images to segmented road maps[29], drawn edges to cats[1], and segmented architectural facades to images of building fronts[79] (see Figure 2.7). A Pix2PixGAN[38] is the conventional approach for image to image translation. It is based on the CGAN where the condition

is the input/source image. As per the original paper, the generator uses a UNet[71] architecture. UNet is a convolutional autoencoder that allows the input image to be converted into a latent vector; matching the input of a vanilla GAN generator. The discriminator is represented by a convolutional classifier defined as PatchGAN[38]. The difference between PatchGAN and a conventional discriminator network is that it only penalises structure at the scale of local image patches. It can be understood as a form of texture or style loss and thus, is suitable for image to image translation tasks. The discriminator is trained by predicting the true pair from the input image and the generator output compared with the input image and the target output. The generator is trained based on the discriminator loss as well as the difference between the output and target image (autoencoder training). The key difference between Pix2PixGAN and other image to image translational GANs, such as DiscoGAN[46] and cycleGAN[80], is that it strictly requires pairwise data. This makes it useful for image segmentation tasks. DiscoGAN and cycleGAN can train on two related and unpaired datasets to produce broader image mappings. For example, turning horses into zebras[58].

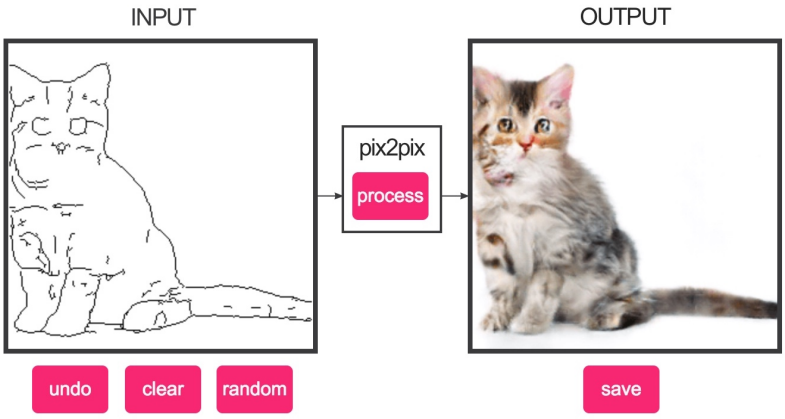


Figure 2.7: Pix2Pix example of converting a sketched cat into a photorealistic representation[1]

Exploring the Latent Space

For communication purposes, the latent space shall be contextualised within an image generation problem. Most of the aforementioned generative methods encode images as a vector. Repeatedly perturbing this vector results in a continuous transition along the learned image space. This can invoke a sense of dymanicness from still image creation and can be harnessed for creative use. In GANterpretations[17], music videos are produced from GAN outputs by traversing the latent space subject to a music recording's frequency spectrogram. By analysing the spectral properties, a direction vector is evaluated at each time step. The vector is then repeatedly interpolated along this direction vector. This has the effect of matching interpolations to music. The latent space may also be explored using latent space arithmetic. If the generator is trained well, latent space arithmetic can produce results which produce highly customisable outputs. For example, in a GAN that produces faces, the direction vector in latent space for a smile may be extracted. If this is added to the latent vector representing a neutral person, the output would contain this person smiling. Additionally, if this was subtracted from the same image, it may produce an image of the person looking sad (see Figure 2.8).

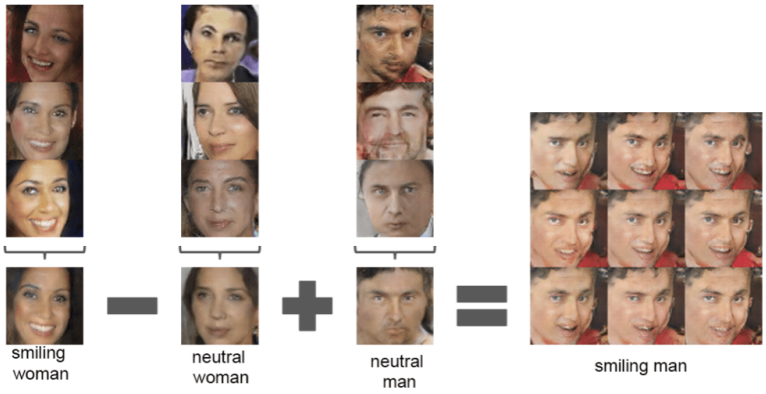


Figure 2.8: An example of latent space arithmetic

2.2 Related Works

2.2.1 Wekinator

Wekinator[26] is an extensive (and at the time groundbreaking) IML toolkit based on a JVM (Java virtual machine). It allows the user to create custom mappings between different formats of data through a GUI. For example, a user could control a musical instrument through custom (learned) gestures detected from a webcam. It also allows the user to plug in other external sensors to interactively control a system subject to learned mappings (which could also be data from other sensors). Wekinator is designed so that there is an element that is the controller and an element that is controlled. A user can supply any controller that can output OSC[28] and any artefact that can be controlled by OSC. Consequently, a significant amount of mapping formats are supported. To learn these mappings, there is functionality for dataset manipulation and real-time model evaluation metrics. A particularly good feature is that the user can reconfigure and explore many aspects of the model subject to input in real time. This is demonstrated in Figure 2.9.

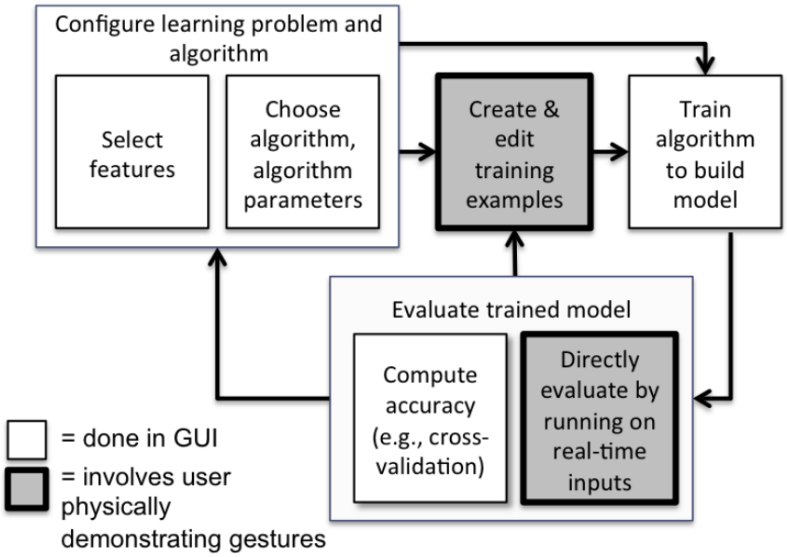


Figure 2.9: The IML workflow supported by Wekinator[26] (source: Rebecca Fiebrink)

2.2.2 InteractML

InteractML[36] is a node based IML Unity3D plugin for designing movement based interaction based on the design principles of Wekinator. It aims to give artists/dancers a greater understanding of ML whilst affording full customisation of the ML model. By utilising a node based system, the plugin supports a creative practitioner's engagement of ML concepts with a high level of education without the requirement of any prior programming knowledge.

2.2.3 RunwayML

RunwayML[77] is a commercial software that allows for easy access to ML for artists. It focuses on professional AI powered video editing (*Sequel*). A secondary software, *ML Lab*, is the most similar of any of the related material to this works aspirations. ML LAB is an end to end ML tool for creatives. It contains a dataset selection/creation step, training/evaluation step, and model output manipulation capability. A good feature of ML LAB is that it requires no installation; it can be run directly through a web interface. It is a phenomenal piece of work that has significantly lowered the barrier of entry for AI artists[73] and it has been hailed for its step forward in democratising ML[72]. There are, however, some limitations to this software. Firstly, the software is not capable of letting the user to train on their local hardware [1]. Whilst this may not seem critical, Runway charges per

[1]The tool, however, let's a user run models on their local GPU but only if they are using Linux.

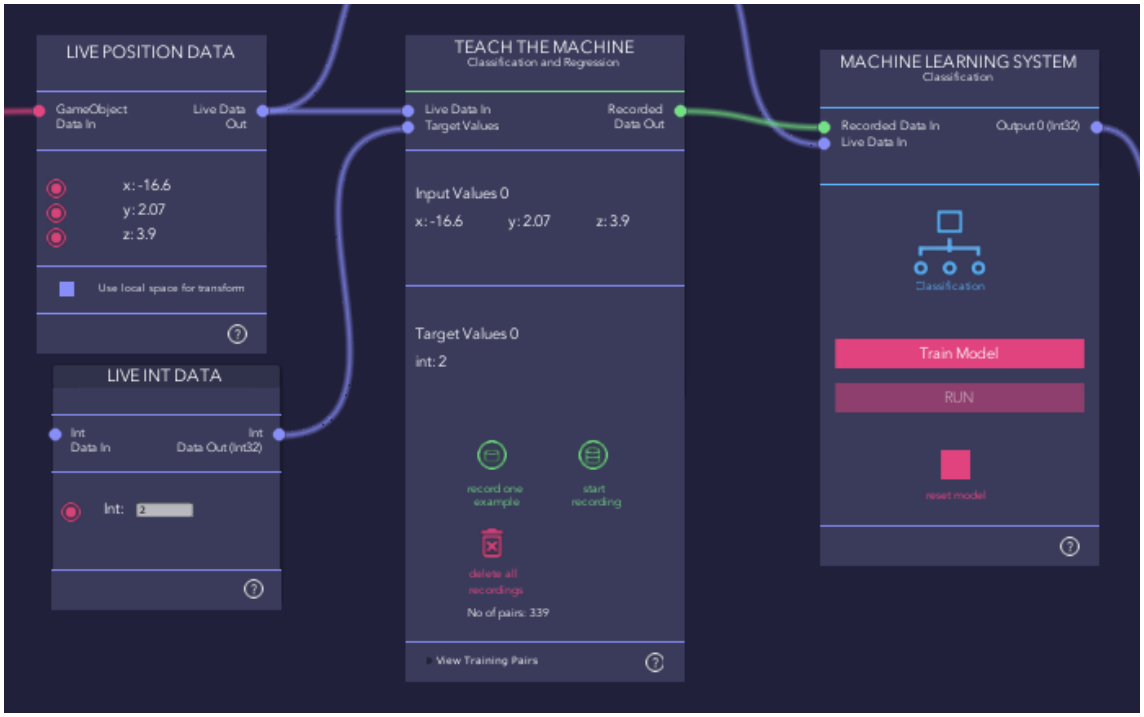


Figure 2.10: Example of InteractML's node based architecture

minute of training (on top of a subscription for this feature), which could incur significant costs for the artist. Secondly, albeit having support to add custom models programmatically, the training GUI only features image generation (GAN), text generation, and object detection (CNN). This introduces difficulty for non-technical artists wanting to use produce works that require image to image translation. Thirdly, the training stage can be improved. Whilst we can see the training process in the format of the current generated image (see Figure 2.11), the user could have more knowledge of the training performance at a given point in time. The only statistics available are the ETA, epochs left, and one loss metric (FID[54] for image generation). Additionally, there is no functionality to interrupt the training process, say if the artist wanted to change a hyperparameter mid-way. Fourthly, the dataset creation tool does not support labelled data (through the GUI). Fifthly, the model outputs may only be manipulated via image-based inputs (ie. not via custom controllers like a MIDI keyboard or sound or via the mouse).

2.2.4 Learning To See - Memo Akten

Memo Akten's immensely popular work "Learning To See"[3] describes an interactive visual instrument that explores bias within ML networks (see Figure 2.12). As an example of IML, the work consists of a real-time representation of webcam input through predictions from an artificial neural network. To construct this, a version of Pix2Pix was trained. The key difference being that the architecture was designed with an unsupervised learning approach. Put another way, the image pairs were constructed from the source image enabling the network to be trained from an initially unpaired dataset. By applying various randomised image augmentations (such as adjusting brightness and contrast) a representative image pair was created. By transforming real-time video frames into this representation, a user can dynamically influence and explore what the network "sees". Critically, Memo created a GUI which allows users to adjust data augmentation parameters of the webcam image data. Important to this work, Memo also created Learning To See in openFrameworks. To accomplish this, he made a first-of-its-kind port from TensorFlow to openFrameworks. Similarly to Wekinator, this was based on a JVM. This work takes heavy inspiration from Learning To See's interactive mechanism and custom Pix2Pix network, with an aim to reduce the amount of dependencies required.

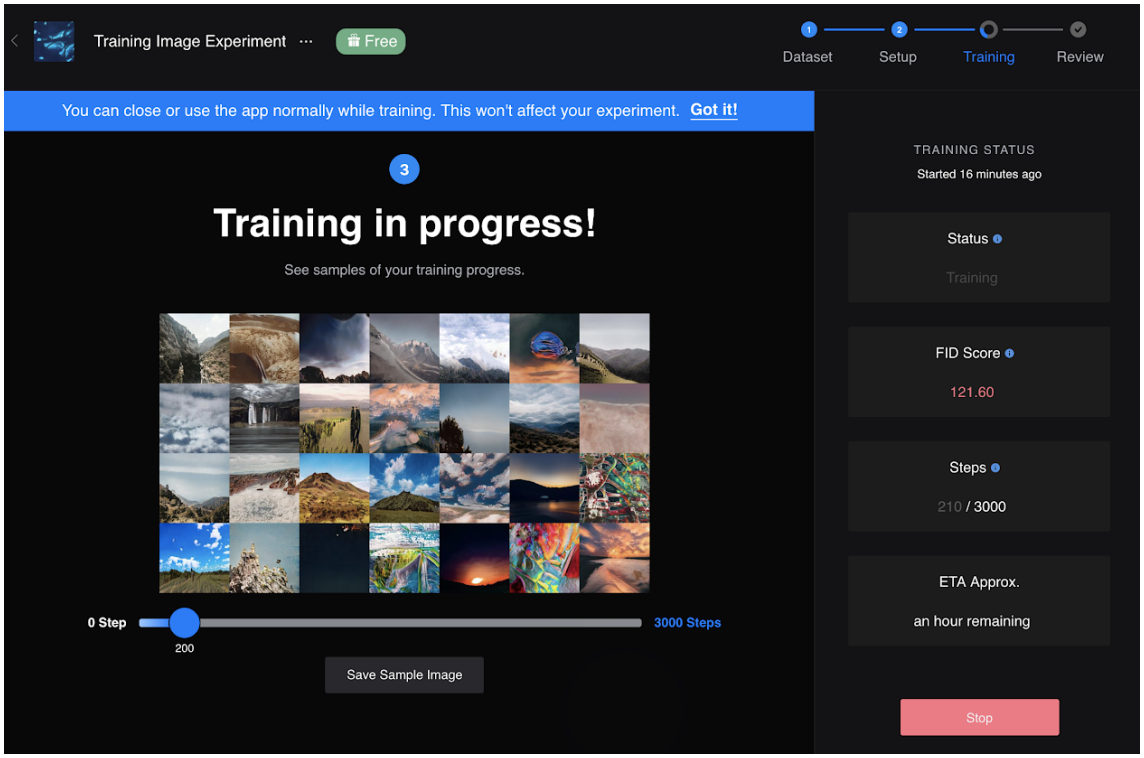


Figure 2.11: RunwayML's training interface

(a) (b)

Figure 2.12: *Learning To See* installation (source: Memo Akten)

2.3 Related Libraries

openFrameworks

openFrameworks (OF) is a cross platform open source C++ toolkit for creative coding. It wraps together many libraries including OpenGL[74] and OpenCV[9]. Because of its emphasis on usability and collaboration, it is one of the most popular toolkits among creatives. OF also contains an extensive repertoire of plugins (OF refers to them as: *addons*) which may be seamlessly integrated into an OF program and can quicken the design process.

ofxTensorFlow2

ofxTensorFlow2 is an OF addon for the Tensorflow 2 (TF) ML library. The addon utilises the TF2 C library wrapped by the open source Cppflow[39] C++ interface. It supports loading TF models pretrained in python and running them inside an OF program.

2.4 IML and Artists

It is with good reason that many of the related works utilise an IML tactic to afford artists the toolset to create with ML. Throughout the literature[6], IML has been shown to support creatives in understanding and deploying ML. One reason being that it has a unique capacity to break a ML model down. As seen, general ML is a multi-step process that represents an input/output mapping. By affording interaction on training data or hyperparameter values, for example, within intermediate points during the training process, the artist is able to control and predict their model output with increased conviction - compared with a conventional "blackboxed" input/output model. Additionally, these steps can impact one another in hard-to-predict ways. For instance, data from one domain might benefit from a slightly higher learning rate compared to data from another domain within the same network architecture. Real time interactivity can facilitate the artist's grasp on the influence of their model's constituent parts. This aids the artist in locating errors and improving their model efficacy as well as, in turn, increasing their learning on the subtleties of ML. In this way, IML reflects the creative process by enabling fast experimentation and continuous iterative design. Finally, IML can enhance a users engagement through interactions on the model output. This helps the artist by providing a more fun and dynamic development process. Indeed, this also affords the production of more meaningful works of art.

2.4.1 User Interface Design for IML

A good IML toolkit necessitates good user interface design. Dudley et. al[25] proposes six key solution principles for the design of an IML interface:

1. Make task goals and constraints explicit
2. Support user understanding of model uncertainty and confidence
3. Capture intent rather than input
4. Provide effective data representations
5. Exploit interactivity and promote rich interactions
6. Engage the user

Chapter 3

The IML Tool

3.1 Overview

The tool currently provides an end-to-end solution for image generation (using Generative Adversarial Networks) and image to image translation (using Pix2Pix) problems. This includes capability for: architecture customisation, dataset curation, interactive training, and various methods of model inference. The software can be viewed as a pipeline with five main scenes: the menu scene, the architecture scene, the dataset scene, the training scene, and the interaction scene. This is illustrated in the navigation bar in Figure 3.2. The user is able to save and load models that have been created with the software. Variables such as the dataset, model architecture, hyperparameters, and status (ie. at what level the model is configured) are all restored in this manner. There is also functionality for the user to load models not trained in the software for transfer learning and/or model inference. It is worth mentioning that the software is still in it's prototyping phase and its design focus is currently placed on the user experience, rather than the user interface.

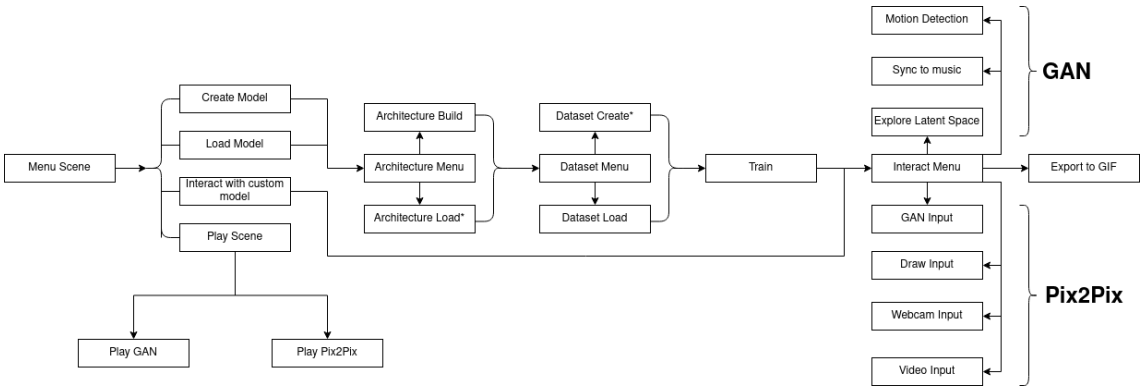


Figure 3.1: The general schematic for the software. Nodes marked with an asterisk were not fully implemented at the time of writing.

3.1.1 Technical Overview

The IML tool is mostly written in C++ using OF. The motivation behind this was due to C++ affording the speed required for dynamic feedback from interactions with the network. Numerous procedures, namely training, were implemented in python and executed within a thread. Each new display instance (other than some subtle state changes) constitutes a distinct scene which is controlled using a custom built scene manager. This improved the productivity of the development workflow as it allowed new scenes to be added more efficiently within the OF framework. A model manager was designed to pass information about the NN model between these scenes. The two managers could safely be accessed through their instances as they both followed the singleton design pattern[56].

Figure 3.2: The navigation bar that appears at the top of the software. It is designed to have a natural mapping[60] of the process illustrated in 2.1. There is a signifier to show which scene they are on and what stage they are at (here: dataset). The user also gets no hover feedback on the buttons that represent the stages that the current saved model hasn't completed.

3.2 Saving/Loading Process

A model is first saved by choosing a valid name and model type. At the time of writing the only model types supported are GAN and Pix2Pix. The software creates a directory with the inputted name and initialises a configuration XML file. This XML file contains all of the model metadata and it is what the model manager manipulates and retrieves information from. A comprehensive list of the metadata is illustrated below. The model directory is also populated with subdirectories: "images" and "saved_networks" which are used to contain per epoch training images and Keras model data (and historical loss data) respectively. To load a model, the model manager reads the configuration XML file from the model name, allowing all scenes that read from this instance to update themselves accordingly. All model root directories are saved under the *saved_models* directory.

- *model_name* - used for model identification - set at initialisation
- *model_type* - set as GAN or Pix2Pix as 0 or 1 - set at initialisation
- *epochs_trained_for* - number of epochs model has trained - set at training
- *dataset_path* - path to the dataset the model used last - set at dataset creation
- *status* - represents what stage the model is up to - set everywhere
- *image_width* - input/output image width - set at architecture definition
- *image_height* - input/output image height - set at architecture definition
- *input_channel* - input/output image channel (3 or 1) - set at architecture definition
- *learning_rateX* - learning rate coefficient - set at architecture definition and training
- *learning_rateY* - negative exponent of 10 - set at architecture definition and training
- *max_epochs* - number of epochs to train for - set at architecture definition and training
- *batch_size* - images per batch at each train step - set at architecture definition and training
- *latent_vector* (GAN) - latent vector dimension - set at architecture definition
- *num_layers* (P2P) - number of layers in generator architecture - set at architecture definition
- *beta* (P2P) - momentum factor for ADAM optimiser - set at architecture definition
- *lambda* (P2P) - L1 regularisation coefficient for generator - set at architecture definition
- *dataset_dir* - used to store dataset to build with - set at dataset creation
- *disc_noise* - noise factor applied to discriminator - set at architecture definition and training
- *random_horizontal* - random horizontal flip? - set at architecture definition and training
- *random_vertical* - random vertical flip? - set at architecture definition and training
- *random_brightness* - random GT brightness factor - set at architecture definition and training
- *random_contrast* - random GT contrast factor - set at architecture definition and training
- *normalise* - normalised generator output? - set at interaction with custom network

3.3 Architecture Definition

After creating a new model, the user is sent to the architecture definition stage. The user can select to build their architecture or load it. The load architecture functionality, however, has not yet been implemented. The load architecture scene is outlined in 6.1. The build architecture scene was difficult to design since it necessitated abstracting and simplifying complex ML concepts. Affording the user too many options may cause confusion - especially among artists without an ML background. Affording too little options results in a lack of capability which may cause frustration - especially among artists with an ML background. A balance had to be struck between the two. With this in mind, the most important parameters were displayed at the top. The less important (optional) parameters were drawn below and only shown if the *SHOW ADVANCED* button was pressed (see Figure 3.5). To further make things easier for artists inexperienced in ML, all parameters are initially set to default values of tried and tested network architectures. Additionally in the intermediary survey (see Section 5.1), multiple participants suggested the software would benefit from hover information about what an element meant. Consequently, help widgets were added next to some of the interface components (see below).

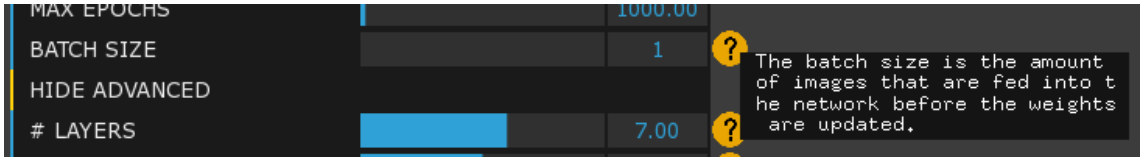
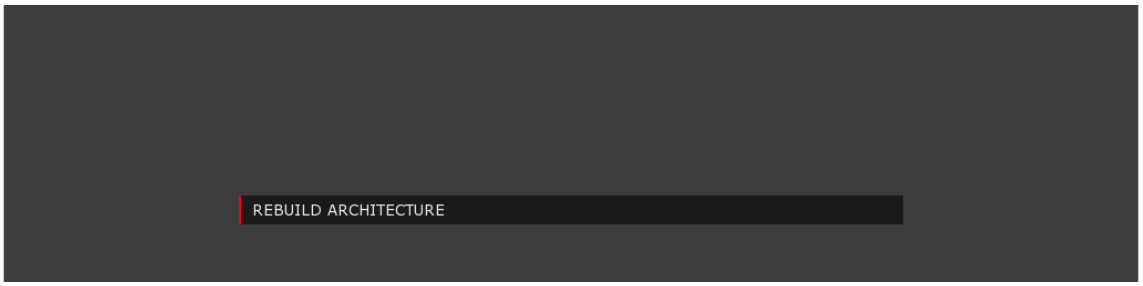


Figure 3.3: The help box appears when the mouse hovers over the help button. It moves with respect to the mouse position. (Note: the cursor was not captured in this image).

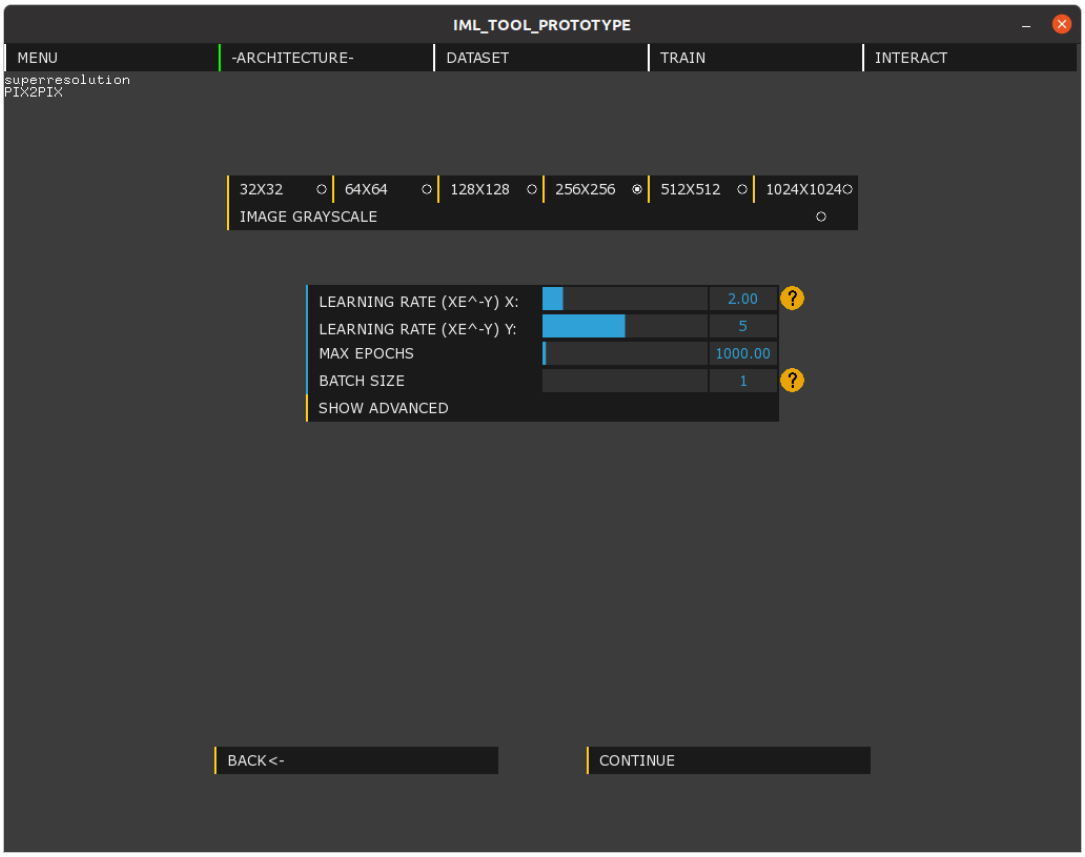
Once the network architectural components have been set, they cannot be changed (without restarting the training process). If they could be changed, Keras would be unable to load the saved model's weights. In this way, if the user navigates back to the architecture scene after the network architecture is defined, they will be met with a different scene (see below). This scene allows the user to rebuild the architecture, but reminds them that this is a destructive action. If *REBUILD ARCHITECTURE* is pressed, they will be met with a final confirmation state which acts as a Lock-In Forcing Function[60].



(a)

(b)

Figure 3.4: On confirm, all data in the *saved_networks* directory is deleted and the model's status is updated to 1 (architecture not built).



(a)

(b)

Figure 3.5: The architecture builder scene (Pix2Pix). Above: the initial scene. Below: the scene after pressing *SHOW ADVANCED*.

3.4 Dataset Configuration

Identical to the architecture menu scene, the dataset menu scene has two options: *BUILD DATASET* and *LOAD DATASET*. The dataset loading/saving mechanic works similar to the model loading/saving mechanic. Datasets are saved under *saved_datasets* in directories with a unique name and are attributed to the model accordingly (the path is saved under *dataset_path* in the configuration file). Initially, the *saved_datasets* directory is empty so there are no datasets to select to load. This can be populated by building a dataset (see Figure 3.8).

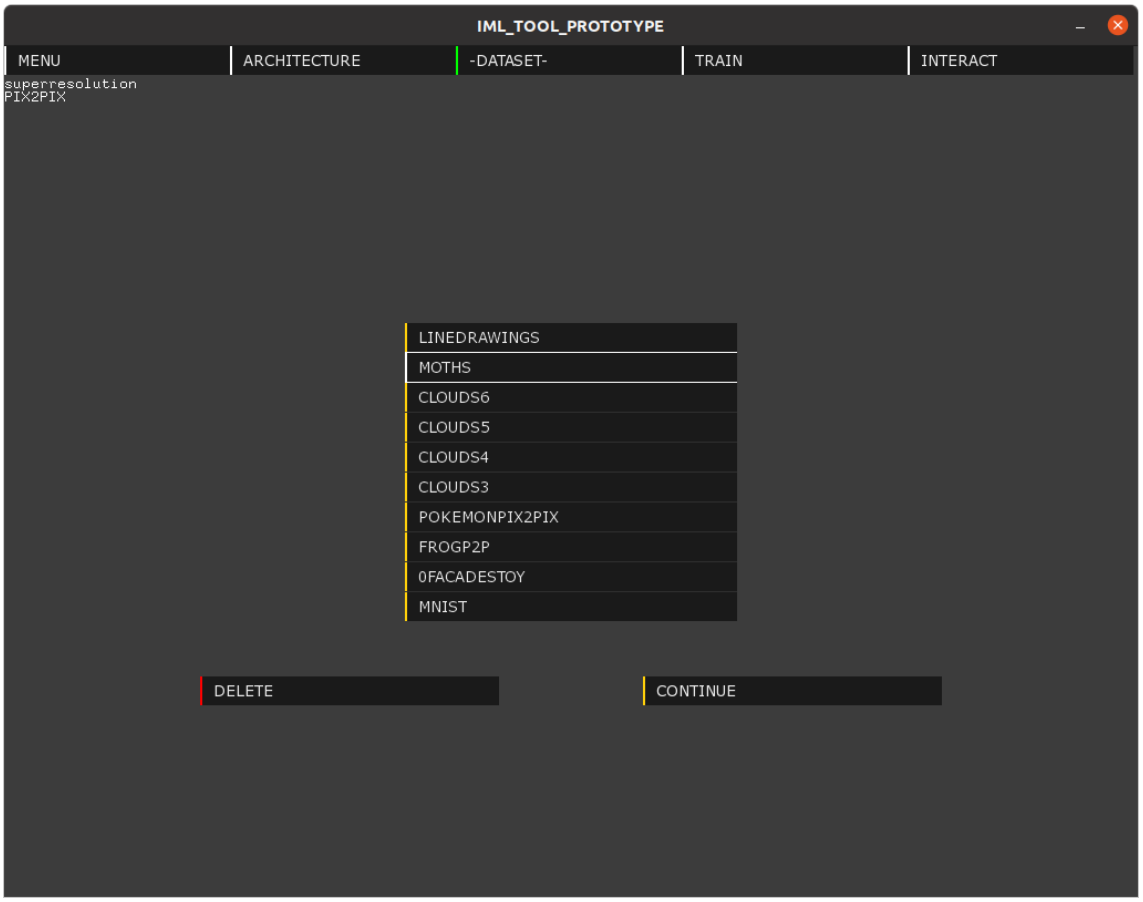


Figure 3.6: The dataset load scene: datasets saved under the *saved_datasets* directory are displayed in a scrollable list.

Figure 3.7: The Pix2Pix image format that the training process accepts. Target output (left) and input (right)

3.4.1 Dataset Building (GAN)

To load a dataset into the software, the user must input a valid name and then select the directory containing the images. The image directory's contents are then copied into the new named directory. Once built, the user is directed to the training scene with the newly created dataset as an attribute. This dataset will now appear in the load scene to be used in future models.

3.4.2 Dataset Building (Pix2Pix)

Due to PixPix requiring a dataset featuring both input and target output images, the software provides more functionality here than GAN dataset building. Linking both the input and output ground truths and organising them into a canonical dataset presented a challenge. To overcome having to collect two separate image files, the python program executed in the training process accepts a dataset of images that each contain both the input and output ground truths. These are interlaced horizontally (see Figure 3.7). The python program then processes and separates the images before training. If the images are already in this format, the user may check the *IMAGES ALREADY PAIRED* toggle, and the process follows identically to the GAN dataset build operation. Otherwise, after a valid name and image directory are supplied, the user may select to build manually or build with image processing. The manual building scene is still under development and is currently left for future work (see Section 6.1).

Building With Image Processing

Inspired by Learning To See (2.2.4), users can curate a Pix2Pix dataset from a directory of unpaired images through image processing techniques. There are three modes of image processing supported by the software, which can be cycled through the *TOGGLE MODE* button (see Figure 3.9). The first mode allows the user to perform grayscaling, inversion, blurring, contrast adjustment, and brightness adjustment. The second mode performs contour detection and the third mode performs Canny edge detection[16]. Each mode affords a level of customisation through an interface. Additionally, the user may press space to cycle through the images in the directory to evaluate the selected parameters. To build the dataset with these augmentations, the user presses *APPLY TO ALL IMAGES*, and the dataset is then created identically as before. Image processing is conducted using OpenCV[8].

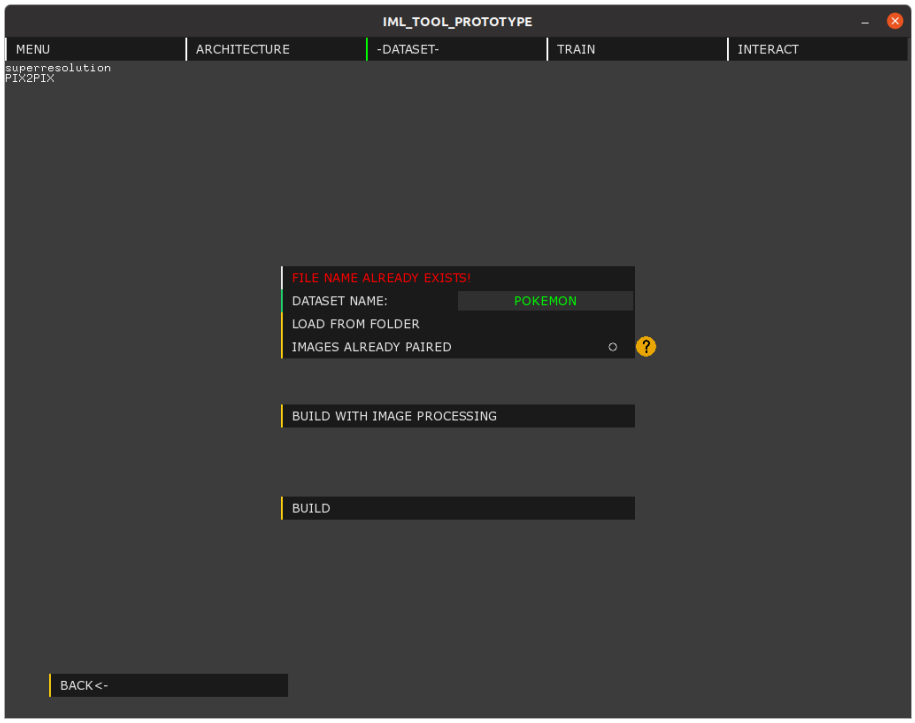
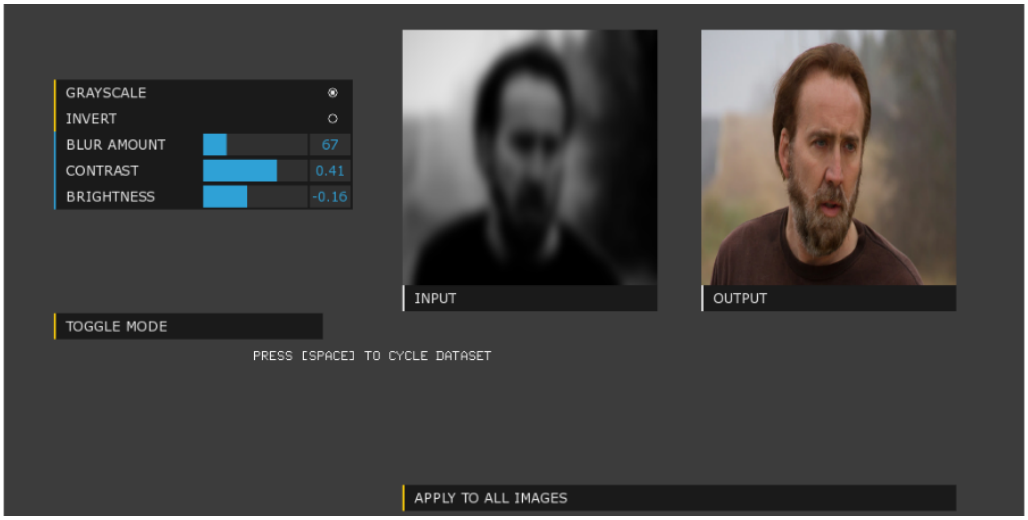
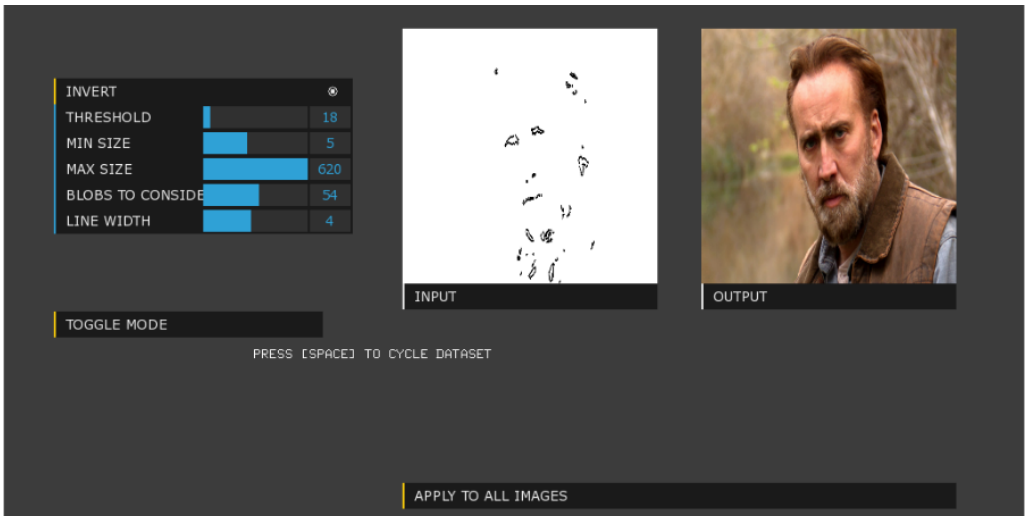


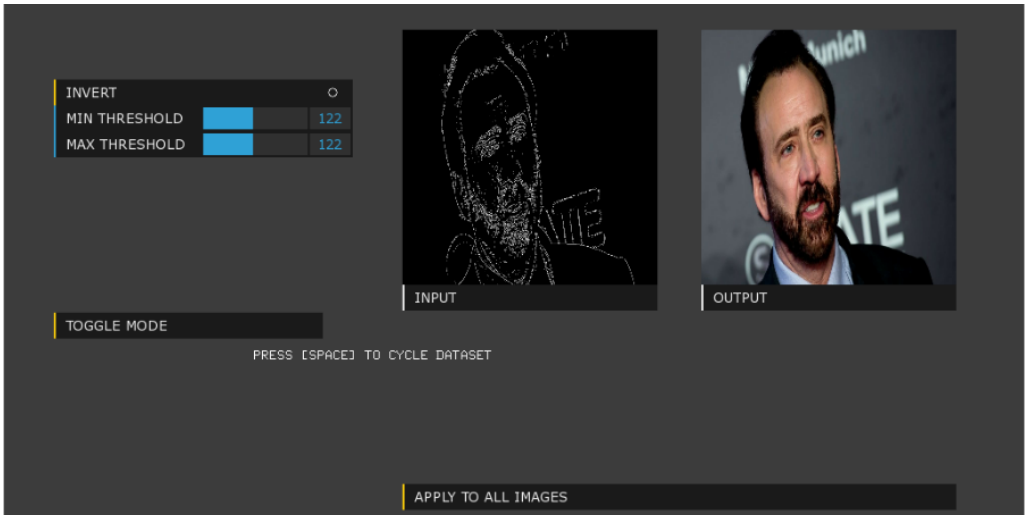
Figure 3.8: The dataset build scene (Pix2Pix).



(a)



(b)



(c)

Figure 3.9: The Pix2Pix building with image processing modes. "Normal Mode" (top), "Contour Mode" (middle), and "Canny Mode" (bottom)

3.5 Training

After the architecture and dataset are configured, the user is permitted to access the training scene. This can can be categorised into two states: training and not training (see Figures and 3.12 and 3.10). In the "not training" state, the user can select to begin/resume training, restore pretrained weights, restart the training process, alter some hyperparameters set during the architecture definition stage. These hyperparameters consist of values that do not disturb the structure of the network (which can only be altered if the architecture is rebuilt). This includes: the learning rate, batch size, maximum epochs, discriminator noise, and the random image augmentations. If the user to selects to restore the weights, they are met with a dialog box instructing them to choose the directory with the desired discriminator and generator models[1]. The user also receives a reminder in the dialog that these models must have the same network architecture as the one configured. This feature gives the user an opportunity to roll back training to a desired epoch or perform transfer learning on a model trained elsewhere. It is a destructive action, however, as it will overwrite the *saved_networks* directory. To ensure the user does not wrongly delete progress, an Interlock[60] protects this operation. The weights may only be restored if the model has trained for zero epochs. In this way, if the model has trained for at least one epoch, the *RESTART TRAINING* button is drawn instead of the *RESTORE WEIGHTS* button. Since the restart training operation deletes the contents in *saved_networks* and sets the epochs to 0, this is also protected with a confirmation message (a Lock-In function).

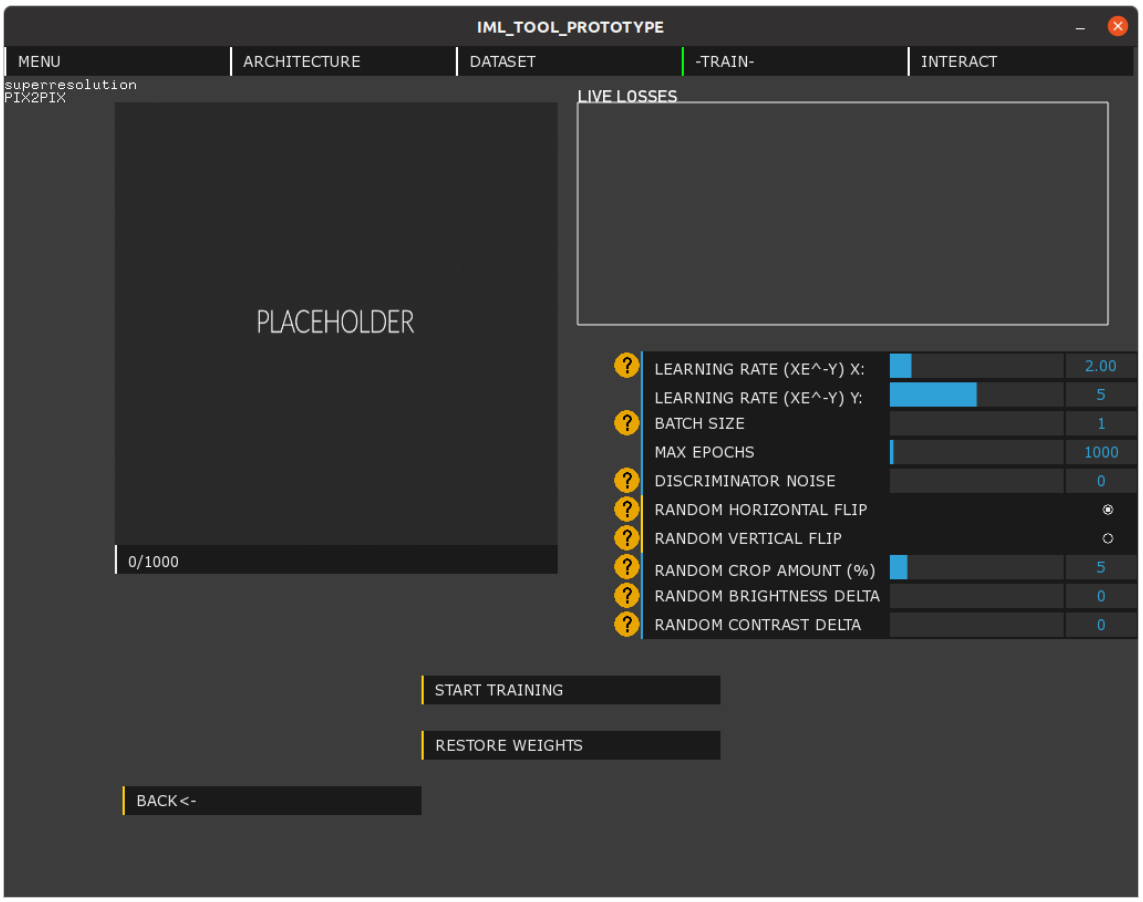
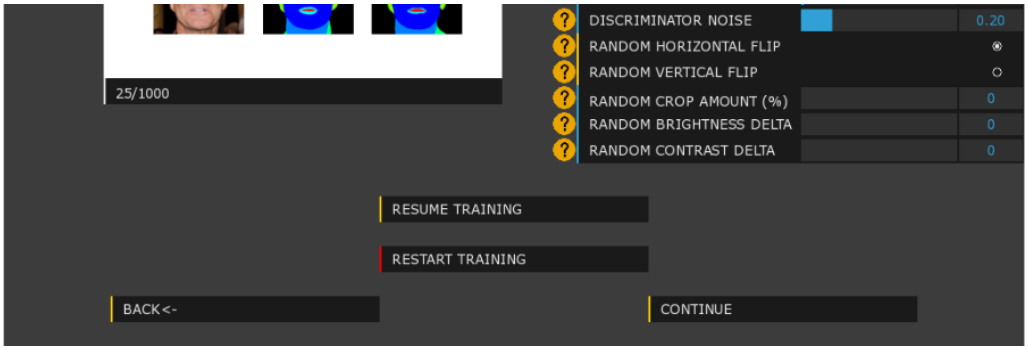


Figure 3.10: The "not training" state in the training scene.

[1]Note: these do not have to be in the *saved_models* directory; they can be from anywhere on the local machine.



(a)

(b)

Figure 3.11: The process of restarting training. The scene also loads the sample image from the last epoch, displays the current epoch, and sets the hyperparameters to those saved on the last run.

When the *START/RESUME TRAINING* button is pressed, the scene enters the "training" state. Three threads are then started. The first spawns a python process (the training thread) which is described below. The second receives messages from the python thread by reading temporary files every eight seconds. The third updates the images and current epoch every ten seconds. The second and third thread could be merged as one thread but the overhead induced by keeping them separate is currently negligible. The navigation bar is also hidden to prevent the user from accessing another stage whilst the model is training. The reasons behind this design decision are twofold. Firstly, it tells the user that training is a computationally intensive task by limiting functionality. Secondly, it removes confusion caused when a user alters model parameters at a separate stage and sees no influence on the training performance. The training thread runs as an independent process that cannot be reconfigured once started. The user is offered three buttons during training: *STOP TRAINING*, *SAVE MODEL AT CURRENT EPOCH*, and *TOGGLE GRAPH*. When *SAVE MODEL AT CURRENT EPOCH* is pressed, the user selects an output directory from the dialog box and the contents of *saved_networks* are copied there under a directory named "{model_name}_{current_epoch}/". The *TOGGLE GRAPH* button cycles through graphs of the generator, discriminator and generator and discriminator losses at each training step (these are generated in the python process). The *STOP TRAINING* button stops all three threads and returns the state to "not training". Considerable thought was placed on how to give the user feedback to show that the model was running. Conventionally, pythonic training procedures print feedback about the network every epoch. Often, especially in image generation networks which require large datasets, epochs can be many minutes or hours apart. Relying on feedback every epoch may lead the user to falsely believe that the network has stopped training when it is betwixt lengthy epochs. To combat this, a graph is updated in real-time (through one of the non-python threads) with loss information on the generator and discriminator networks at each training step. This has the interesting side-effect of "flatlining" if the operating system kills the training process[2]. Additionally, because the training process requires time to initialise before training begins (and losses are updated), information on its status are also displayed to the user.

[2]This can happen if the network is too computationally demanding for the machine and not enough memory can be allocated

Figure 3.12: The "training" state in the training scene. The live loss graph is being hovered by a missing cursor (top right). The loss graph, when toggled, indicates which colour is the discriminator (blue) and the generator (red) through a heading on their respective graphs (middle right). The text updates at each epoch and also feeds information to the user during initialisation (middle left).

3.5.1 Training Thread

The training thread conducts an `execv` system call on a python file for training. The python file chosen depends on the type of network. At the time of writing, this constitutes separate files for Pix2Pix and GAN training. The python file parses arguments supplied in the system call to manipulate the desired training configuration. This includes: architecture parameters, training hyperparameters, and dataset specification. Arguments outlining where to output certain data are also included. Architecture arguments (such as number of layers) are handled in a class. Using these arguments, the class selects from predefined (tried and tested) network structures[3]. Dataset arguments (image dimensions and dataset path) were particularly easy to handle. In TF 2.6, a dataset may be created from a directory of images. These images are then resized according to the arguments without affecting the dataset on the disk. Crucially, this means the user does not need to create a new resized dataset for models requiring different dimensions. The ADAM optimiser is used for both Pix2Pix and GAN training. The loss functions used were taken directly from the original Pix2Pix and GAN papers[38][32]. The random augmentations are performed before each training step. For Pix2Pix these are seeded every step and carried out statelessly to ensure the same augmentations are applied to the input and target output[4].

[3]Network structures were selected from sources such as the TF documentation and Github.
[4]The input in Pix2Pix training is not affected by random brightness or contrast adjustment

Communicating With The Main Program

Originally, the python process and the main software communicated through named pipes. After the information often came out garbled, however, data was sent and received through temporary text files instead. This was a simple transition since a pipe (on a POSIX system) is just a special instance of a file. The root directory for the temporary information files are passed as an argument to be (over)written in the python process and received from the main program. Live losses and status updates are transferred in this manner. The images per epoch are saved according to an image root directory argument. The networks are overwritten and saved every epoch according to a checkpoint directory argument, with the latest epoch at the beginning of the file name. The main program processes this file name to get the current epoch.

3.6 Interaction

It was critical that the software offered rich interactions with trained networks. This was vital so users could have a variety of forms to express their models and export them in a format that met their creative needs. Perhaps more significantly, however, it can be noticed that an evaluation scene is missing from the software. This was identified as a crucial step in Section 2.1. By providing interactions that allowed users to sufficiently explore a models behaviour and output space, this would eliminate a need for conventional evaluation metrics. In turn, the interaction scene constitutes the evaluation stage. Pix2Pix and GAN models have different interaction capabilities. They both, however, support the simplest interaction which involves exporting the images saved every epoch in training as a GIF. The user selects an output folder and presses the *BUILD* button. This executes a python file with supplied arguments similar to the training thread. Finally, each mode that performs dynamic/real-time interaction has the capacity to record video of the ML output and/or take pictures[5].

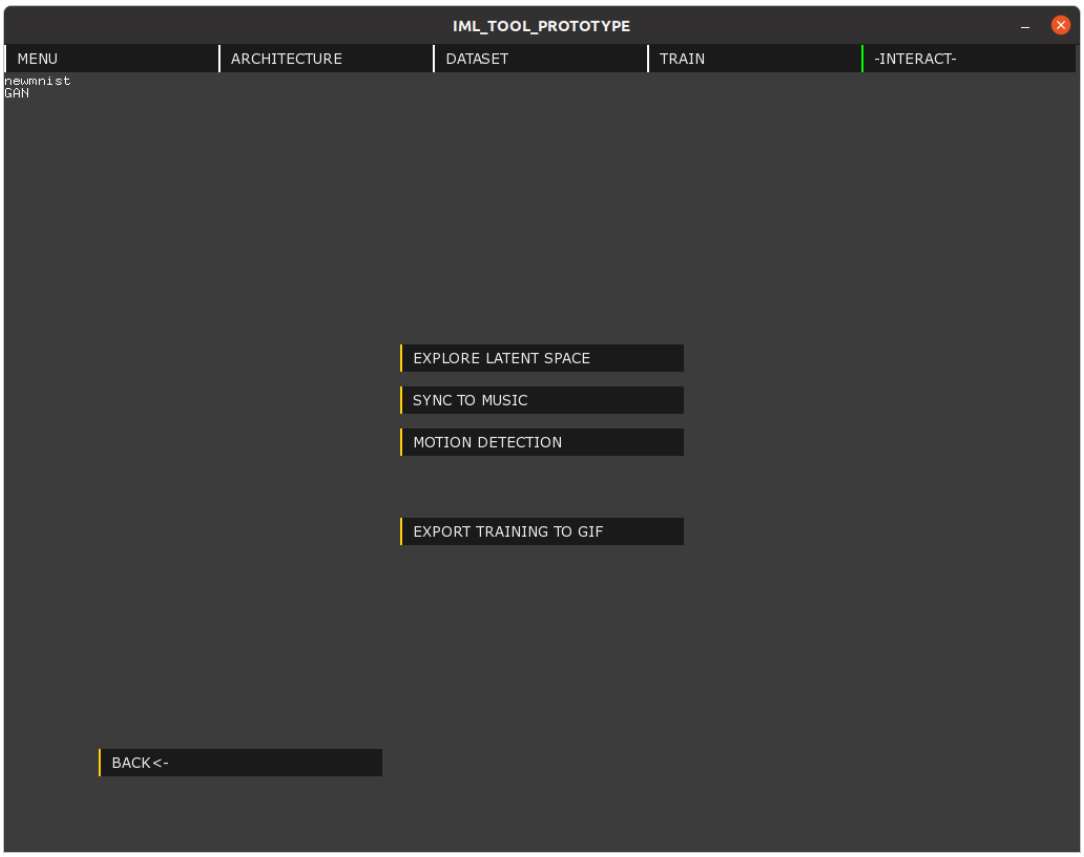


Figure 3.13: The interaction menu scene (GAN)

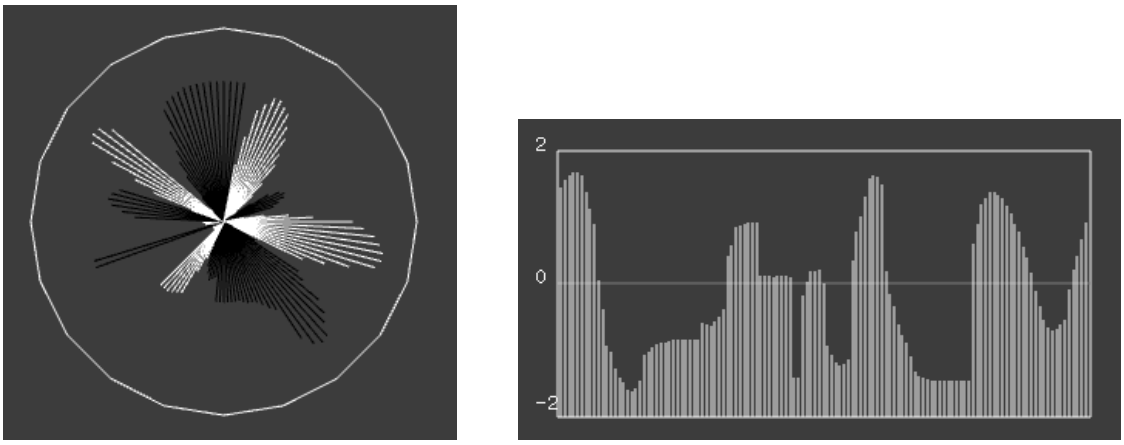
[5]Whilst the model is always drawn a fixed size, the exports are the true network output (no up/downsampling).

3.6.1 GAN Interactions

The GAN interaction menu is illustrated in Figure 3.13.

Exploring The Latent Space

The purpose of this scene is to allow the user to control the output of a GAN by manipulating the latent vector. An interface had to be designed that illustrated and interacted with, effectively, a long list of numbers. Two interfaces were designed each with a different functionality (see Figure 3.14). The dial widget represents each value of the latent vector radially. The user can "rotate" the values which has the effect of translating each values index subject to the rotated angle. This is accomplished by dragging and releasing the mouse. Guide lines are drawn to help the user determine the rotation amount. The second widget illustrates the values along a Cartesian plane where the y component represents the value and the x component represents the index. By dragging the mouse over the graph the user can set the value subject to the mouses relational y coordinate. The program limits the latent vector to 2 and -2 since most GAN implementations train using a random normal distribution with mean 0 and standard deviation 1. Therefore, the range 2 and -2 covers 95% of the trained latent space.



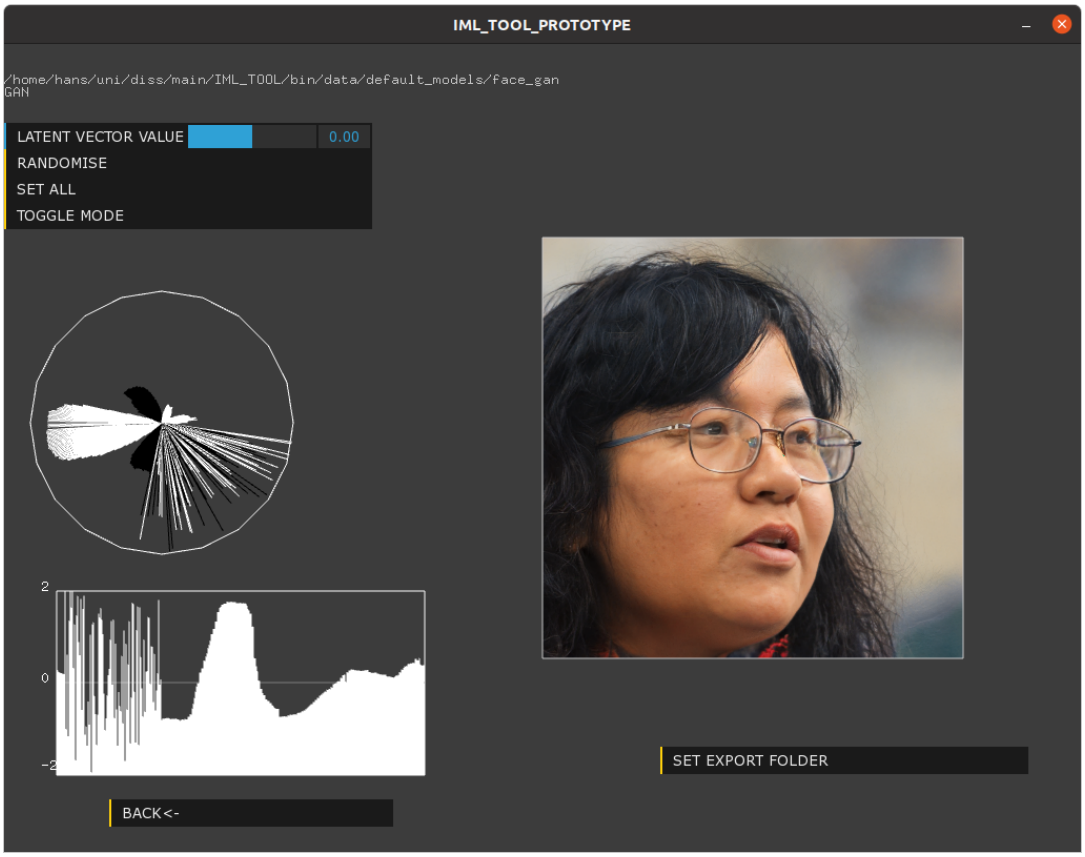
(a) (b)

Figure 3.14: The dial widget (left) and the graph widget (right) representing the same latent vector of dimension 128. Lines in the dial widget are drawn black if they represent negative values.

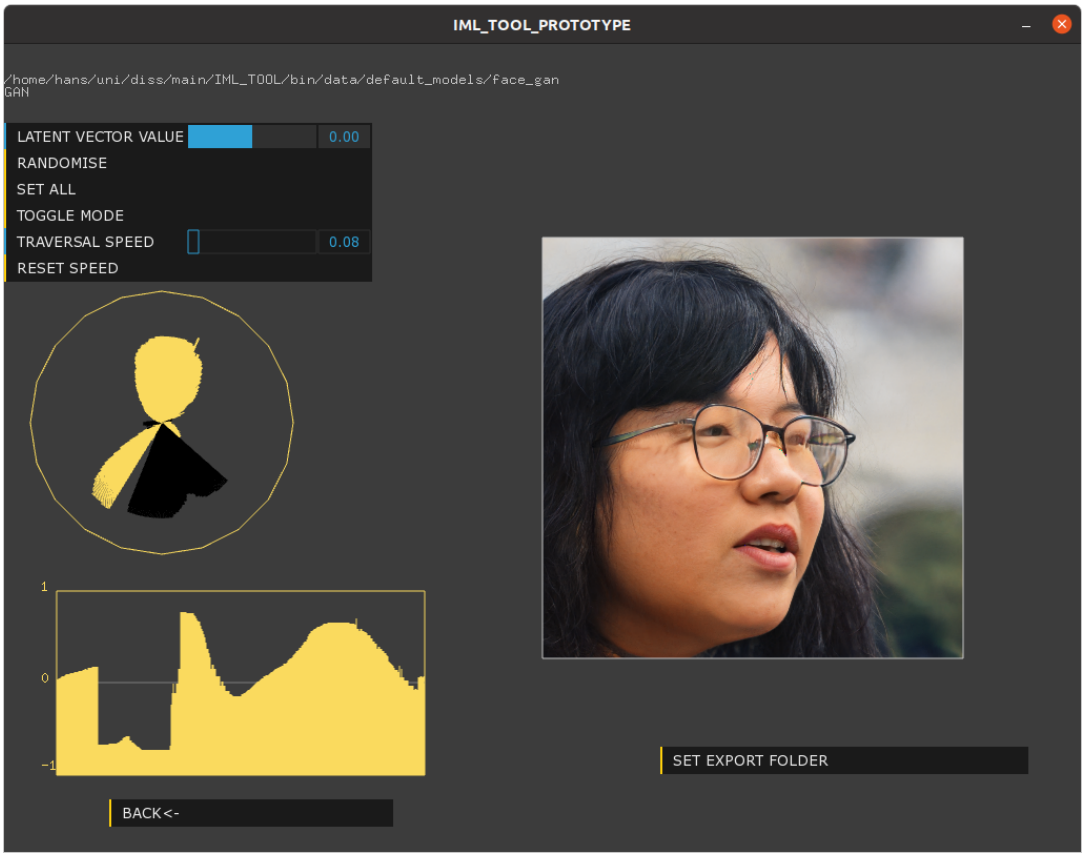
There is also capability to randomise the latent vector, set all of the latent vector to a value (controlled through a slider), and automatic traversal. Automatic traversal can be performed after the user presses the *TOGGLE MODE* button. The widgets now represent the perturbation or "velocity" vector and are drawn in a different colour to reflect that. The speed may also be controlled through a slider. Traversal works by adding element-wise the latent vector and the velocity vector, which is scaled with the speed parameter.

Sync To Music

By uploading an audio file, the user may sync GAN output through latent vector interpolations to the audio. After configuration, a thread executes a python program that produces the video. The program is a wrapper for the Lucid Sonic Dreams[4] module. Similar to GANterpretations[17], the module converts the audio into a specotogram and preprocesses the interpolations for each frame. The interpolations are then looped to produce individual frames. These frames are stitched together into a video and saved. The interface accepts numerous arguments, such as the pulse strength, motion strength, and the frames per second. Since this is a lengthy process, the user is provided with regular feedback on its execution status. There is also an option to kill the process through a button similar to training. The thread will detect when the process is complete and return the user to the music sync configuration scene.



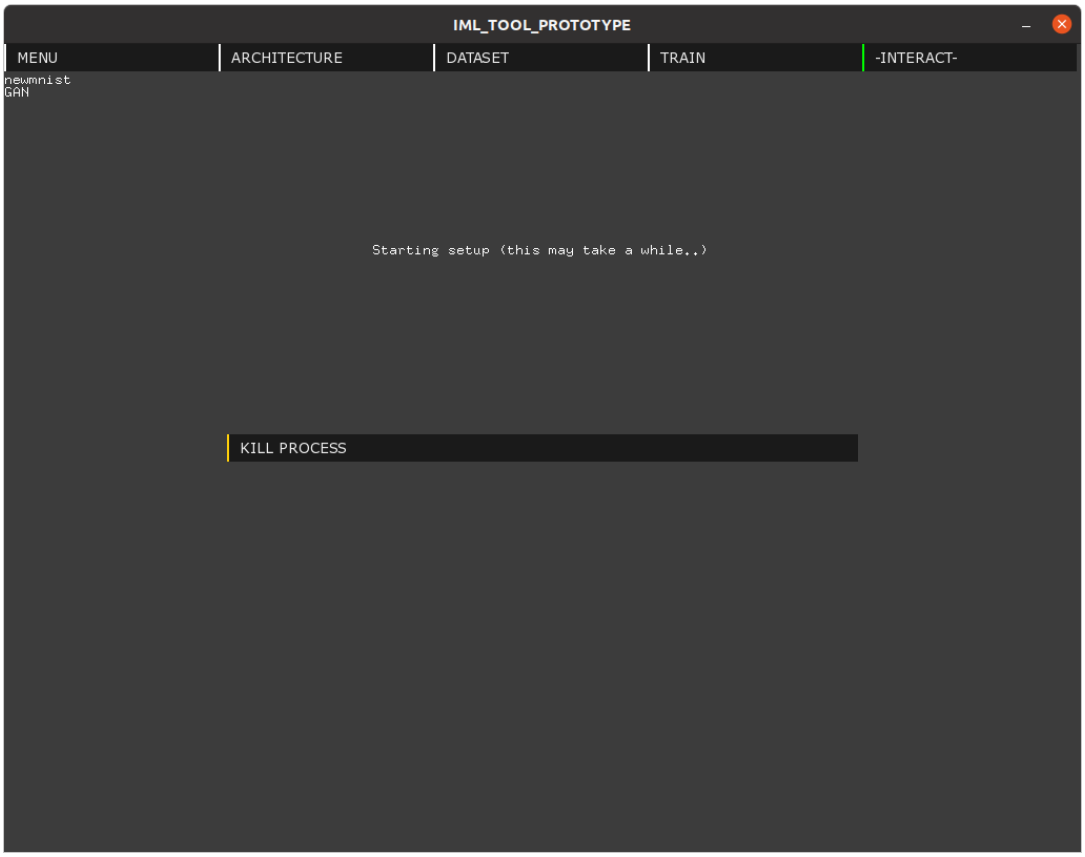
(a)



(b)

Figure 3.15: The explore latent space scene. Manipulating latent vector directly (top) and manipulating the "velocity" vector (bottom).

(a)



(b)

Figure 3.16: The sync music scene. There is a show/hide advanced button on the sync music configuration scene (top). Regular status updates are received and displayed after running (bottom).

Motion Detection

This interaction controls the latent vector through real-time webcam data. There are two modes: frame difference and non-frame difference. Frame difference alters the latent vector subject to the amount of change between adjacent frames. Non-frame difference only takes the current frame into account. The mode is selected through the *FRAME DIFFERENCE* toggle. Latent vector values are assigned by randomly sampling the same quantity of pixels from the webcam data as the latent dimension. Consequently, the sampled pixels are one-to-one mapped to an element in the latent vector. These pixels can be resampled through the *RANDOMISE* button. In the non-frame difference mode, the pixel value is grayscaled and normalised to a range of -1 and 1. This is directly set as the new latent vector value. In frame difference mode, the pixel-wise difference between the last grayscaled frame and current grayscaled frame is calulcated. Each pixel is then normalised to a range of 0 and 1. This is then multiplied by a random perturbation vector (set when the latent vector is randomised), with the same dimension as the latent vector. The value is then added to the corresponding latent vector element. If the latent space is particularly sensitive (such as with large GANs), the user may control the strength in which the operations are performed through a slider. The strength parameter scales the normalised vector value. However, the latent vector will remain in the [-2,2] limit.

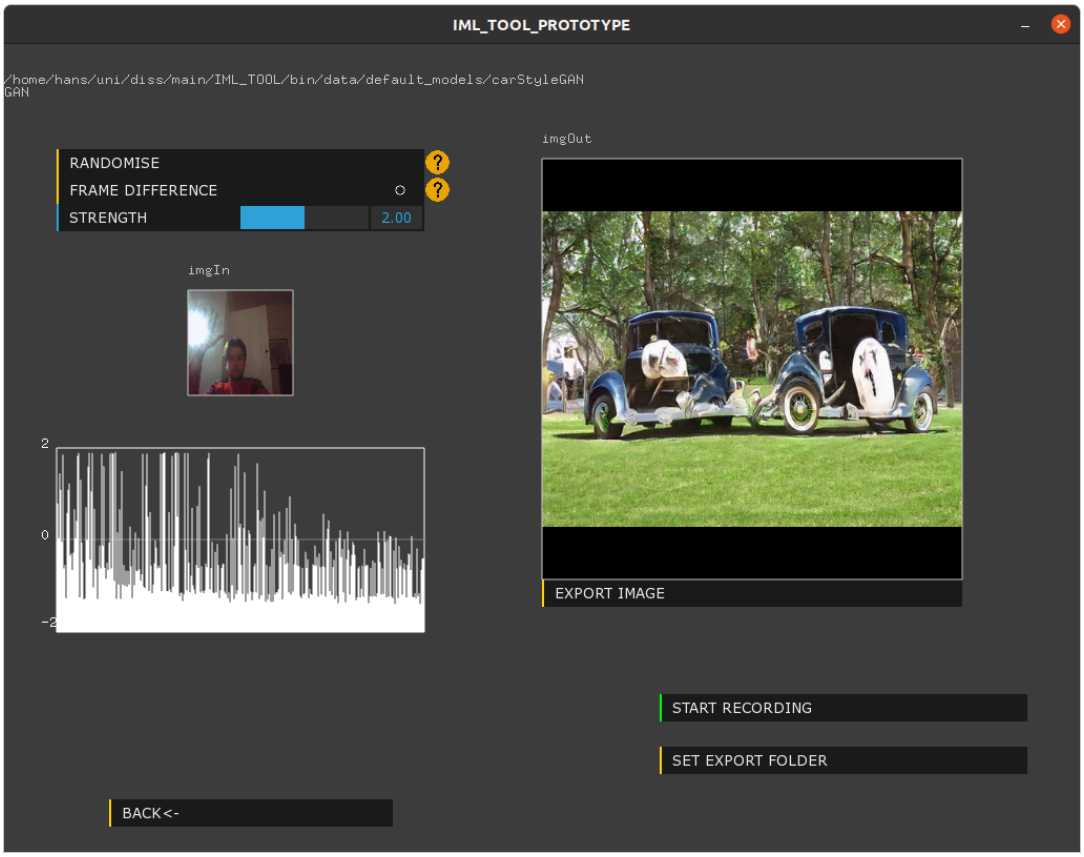


Figure 3.17: The motion detection scene (GAN). This also demonstrates the state that appears when the *SET EXPORT* button has been pressed (and a folder has been selected) in a live feedback interaction scene.

3.6.2 Pix2Pix Interactions

The Pix2Pix interaction menu is illustrated in Figure 3.18.

GAN Input

There is capability for the user to use the output of a previously trained GAN as input into a Pix2Pix model. Thus, a Pix2Pix network may be controlled identically to a GAN in the explore

latent space scene. Before the network can be controlled, the GAN network and its configuration must be specified. In this preliminary state, the user selects the generator's directory (from a system dialog box), the input latent vector dimension, and the output image dimensions (including the image channel). After continuing, an adapted explore latent vector scene is displayed (see Figure 3.19). The GAN output is automatically up/downsampled to the input pix2pix dimensions.

Pix2Pix Input

The software also supports a separate Pix2Pix network's output as input into a Pix2Pix model. Similar to the GAN input scene, the user must configure the network before they can control the model in real time. The input Pix2Pix network is controlled using the webcam in a reduced version of the Pix2Pix webcam input scene (see Section 3.6.2). The user may apply some basic image processing on the webcam input data including blur, contrast, and brightness adjustment (see Figure 3.20). The Pix2Pix output is automatically rescaled to the specified input dimensions.

Draw Input

In this scene, the user is able to submit images or draw input in real-time. To use an image as input, the user can simply drag an image file into the input window (which can be drawn over). For drawing, there are three states (see Figure 3.22) that can be toggled a button. The first state illustrates the available keyboard shortcut commands. The second state tells the user what colours are currently in the palette. The third state provides an interface to draw with a custom colour and set the brush radius without a keyboard command. Since this scene required the most precise input from the user, it was paramount that operations where the mouse had to be directed away from the input window were minimised. This motivation resulted in an extensive list of shortcuts which covers all operations except leaving the scene, saving/loading a custom palette, and setting an export folder. The input window is also much larger compared to the other interaction scenes because of this reason. By including a colour palette, the user may efficiently cycle through the colours that fit the networks input image specification. This is especially important for colour encoded inputs (such as in Figure 3.7). Furthermore, an option to save and load custom colour palettes is implemented which may be cycled through the C/V keys.

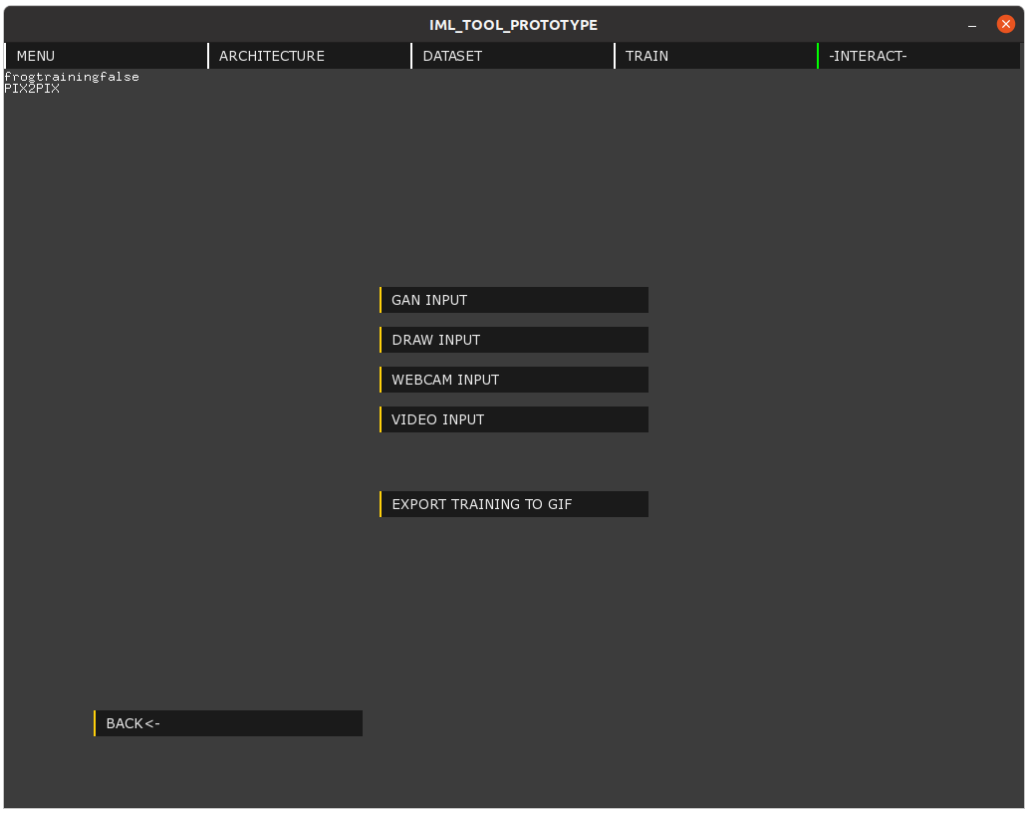


Figure 3.18: The Pix2Pix interaction menu scene.

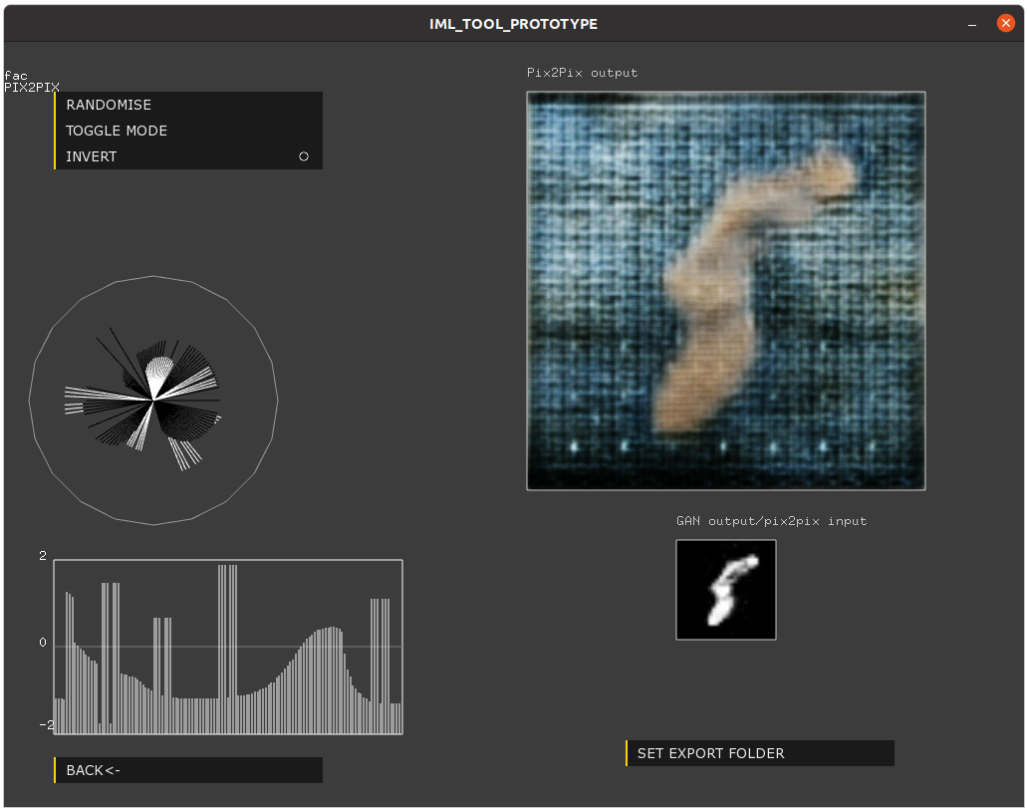


Figure 3.19: The GAN input interaction scene (P2P) - after configuration.

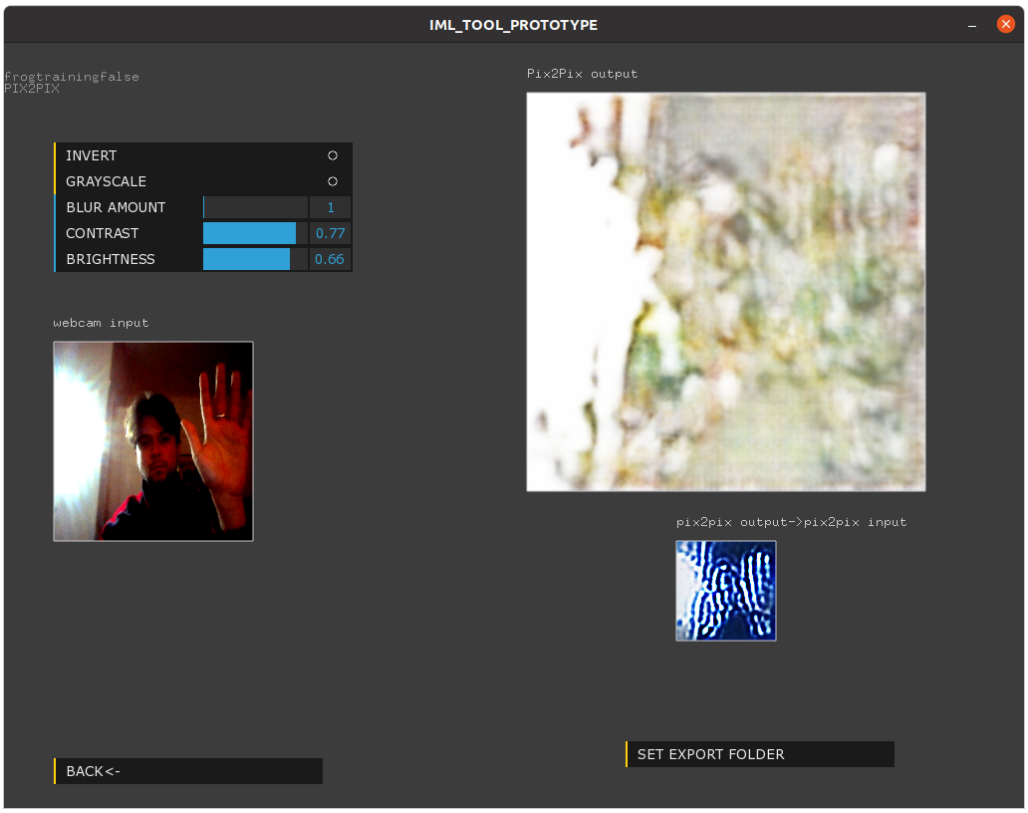


Figure 3.20: The Pix2Pix input interaction scene (P2P) - after configuration.

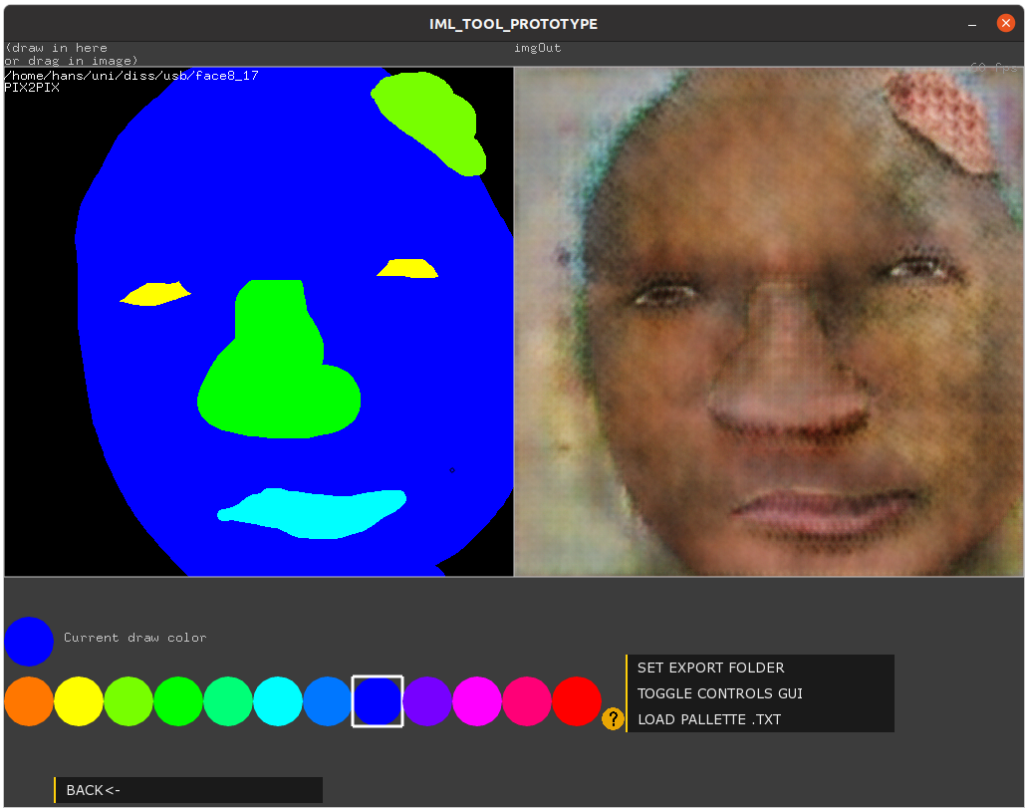
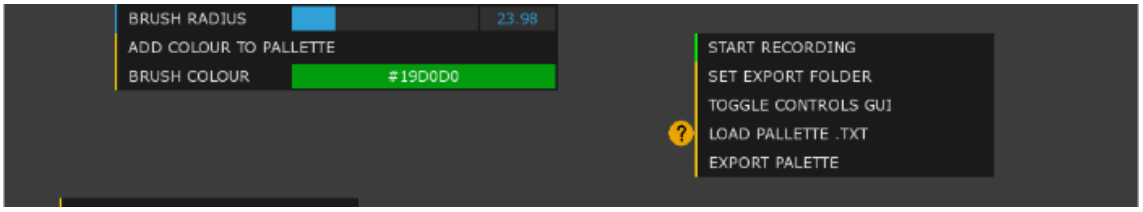


Figure 3.21: The drawing interaction scene (P2P) currently in the "colour palette" state.

(a)



(b)

Figure 3.22: The drawing interaction scene (P2P) "keyboard shortcut" state (above) and "manual interface" state (below). As opposed to a brush, drawing boxes is also supported and is toggled with the D key. Colours can be selected through a colour picker interface on hover.

Webcam Input

Live webcam data can be inputted into the network with three modes of image processing using OpenCV[8]. In the first mode, the video feed can be grayscaled, inverted, and custom adjustments can be made to the brightness, contrast and (Gaussian) blur amount. The second mode determines the contours in the webcam data. Parameters that affect contour detection, such as the minimum and maximum contour size, can be adjusted by the user. Additionally, the user can customise the contour's line width and invert the output. For more accurate detection, the background image can be set (either through a button or the B key). In this instance, contours will be determined through

the difference between the specified background image and the current input frame. The third mode performs Canny edge detection[16] on the webcam data, also with adjustable parameters (see Figure 4.8).

Video Input

Similar to the music sync scene for GAN interaction, this functions with a preliminary configuration state followed by a state displaying status information from a spawned python thread. The python program accepts arguments describing the input and output video directories as well as model configuration data. It passes each frame to the network and merges the output frames before exporting the new video.

3.6.3 Custom Model Interaction

It was important not to limit the interactive functionality to just models created and trained within the software. This was for numerous reasons. Firstly, users with very little technical experience might be discouraged from using the software. Even though the program attempts to enhance understanding of the ML development process, it does not remove the basic order of operations required to produce a trained generative network. These operations remain to demand a level of understanding which has the potential to intimidate the user. Such a user might prefer to download a pretrained network instead of pursuing the learning process. Secondly and contrarily, users with a strong technical/ML background might also be discouraged. They will likely have an established ML development workflow and might prefer to create the network externally, but still want to utilise the software's interactive features. Finally, model development - specifically training - is often a lengthy procedure. During which, users are prone to becoming disinterested. By reducing the amount of time to reach the most dynamic features, the software can keep users engaged (first time users especially). Custom model interaction can be accessed through the *INTERACTION* button in the menu scene. There, the user selects the type of model to be interacted with. At the time of writing, these options are an image to image translational model and an image generation model[6]. The user is then instructed to select the pretrained network directory followed by its configuration. For example, for the image generational option this includes the input latent vector and output image dimensions. Since some networks may produce a normalised output, the user can specify this with a toggle. On continue, the interaction menu scene is displayed. The scene is identical to the "original" menu, only without the navigation bar. The program identifies a custom model by setting the status in configuration to an arbitrary negative value in the model manager. A custom model has access to all interactions except exporting the training images to a GIF.

3.7 Other Features

Play Scene

Whilst custom model interaction efficiently attempts to sustain engagement, configuration is still essential before a user can reach interactive elements. To minimise the amount of clicks to produce an output, the software includes a play scene which can be accessed from the main menu. The play scene features interactions on two pre-trained models. At the time of writing, these are the drawing interaction on a Pix2Pix face generator (see Figure 3.21) and the latent space explorer interaction on a GAN trained on car images. This has the added benefit of allowing users to play with the software's functionality without any prior technical knowledge.

Plug and Play Software Design

By outsourcing most of the ML development process to python programs, it is particularly easy for a user/developer to customise how training is performed. For example, a user/developer could provide a custom architecture such as StyleGAN[43] or a variational autoencoder [48]. This would be accomplished by refactoring the original python files, or substituting the files entirely. For this to be seamlessly integrated with the software, the python programs must accept and correctly

[6]These are purposefully not written as Pix2Pix or GANs. This is because, in practice, any image to image model may be loaded (as long as it is configured correctly) such as an autoencoder. Similarly, any image generational model that takes a latent vector as input may also be interacted with.

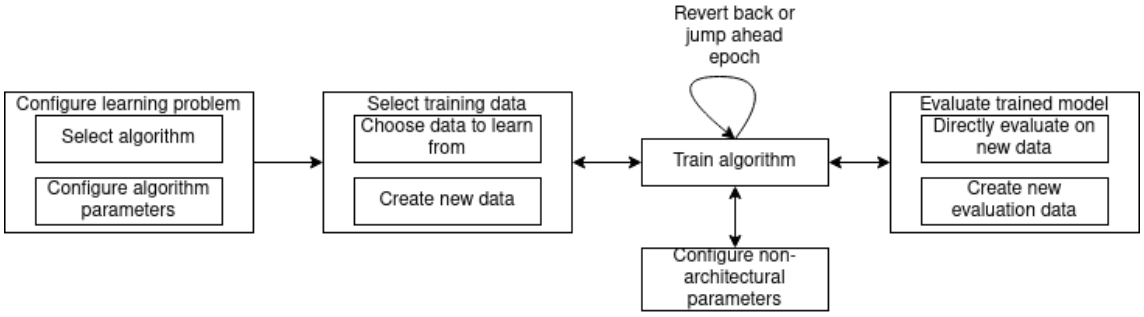


Figure 3.23: The IML workflow provided by the software.

process relevant arguments and output information according to the specification outlined in section 3.5.1. To make this transition easier, the supplied python files are partitioned according to individual operations, which serves to increase re-usability. For instance, a separate file contains a `LossManager` class which handles producing loss graphs and another file handles architecture definition. Such a design provides the scope for the ML artist community to produce and release their customised python files, which could be utilised by less experienced ML artists. Eventually, there is plans for a python module to be created to make this process easier (see Section 6.1).

3.8 Planned Installation Protocol

At the time of writing, there is no universal installation functionality. The software was written inside the linux operating system so the transition to MAC OSX will not present many difficulties, as they are both based on UNIX. Windows, on the other hand, is based on DOS. Since the software is built upon C++, which is a low level language, complications may arise when porting to Windows due to broken dependencies. To combat this, a Docker[57] container will be initialised with the Linux OS used in development (Ubuntu 18.04)[7]. A bash script executed inside the Docker container will install CPPFlow and TensorFlow 2 C API. This will help minimise dependencies for the majority of the program. Additionally, all python code was written inside a virtual environment. This means that all python dependencies can be accounted for by initialising a new virtual environment (in the container) and installing (through pip) all of the python requirements from a text file. At startup, a separate bash file will be executed. This will activate the virtual environment and include the Tensorflow API path in the `LD_LIBRARY_PATH` environment variable. The software can then be run normally.

3.9 IML User Experience

The software and it's IML pipeline (see Figure 3.23) can be summarised as follows: After the ML algorithm is selected (Image to Image or Image generation) and architectural parameters are configured, the user may interact with any of the later stages. For example, the user is able to train on a dynamic dataset that can be appended to, or replaced entirely, without requiring a complete restart of progress. Between epochs, the user can also directly evaluate the model (through the interactions described in the previous section) and select new training parameters such as the learning rate and random image augmentations. The user can also save an instance of the model on the local machine and load previously saved models throughout training. Different stages can be accessed either through back and continue buttons, or through the navigation bar (which indicates the stage that the user is currently on). Stages cannot be accessed if the user hasn't reached a state of configuration in a previous stage. For instance, the user isn't permitted to begin training if a dataset hasn't been configured.

[7] There will also be an option to install a GPU version of the software. This will utilise `nvidia-docker` instead and organise CUDA[61] installations (`https://docs.nvidia.com/cuda/wsl-user-guide/index.html`). Furthermore, it will install the GPU version of the Tensorflow 2 C API.

Chapter 4

Works

This section demonstrates the software's artistic capacity by describing works that were personally created using the tool.

Digital Identities

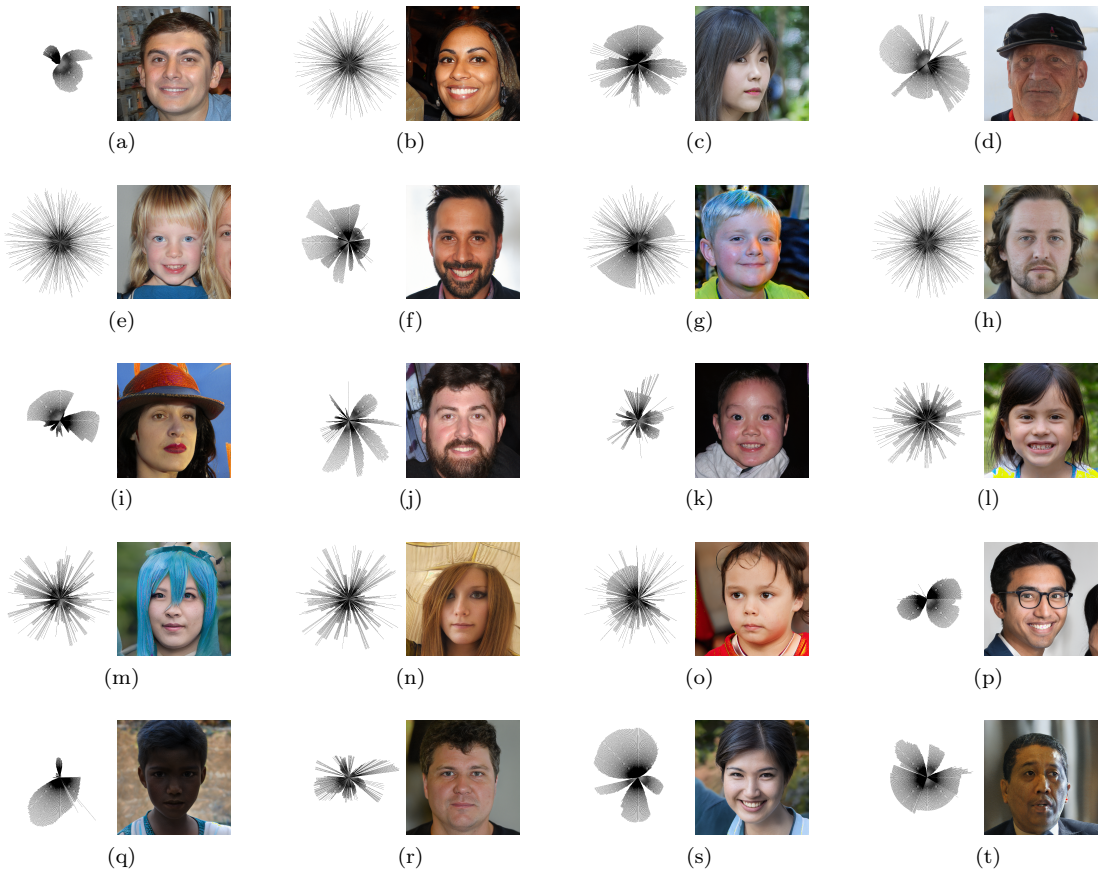


Figure 4.1: A sample of images from Digital Identities.

Digital Identities is a series of images consisting of artificially generated faces and the radial representation of the latent vector that was used to generate them. It aims to demystify and unveil images created using artificial intelligence by revealing the true identity of their form. Thus, the observer is reminded that these faces are deceiving and false; the faces belong to nobody. It demonstrates that the latent vector and the face are equivalent - they are both but an expression of numbers. This provokes thought surrounding the power of artificially intelligence. How does anything produce photo-realistic faces from a series of numbers? Should we be fearful of this

power? A power that not many understand. How could a human ever interpret those numbers in this manner? A language spoken only by machine. To me, this invokes sinister connotations. If artificial intelligence isn't properly regulated and understood, it may start speaking in a language that humans cannot decode.

The generative network used was a StyleGAN pretrained on a dataset of 70,000 faces (FFHQ[43]). The latent space was then explored in the software and the radially represented latent vector was captured along with the related face[1].

Neural Frogs

Neural frogs is an ongoing work that experiments with the extent in which the artist can be removed from the generation process. A network that generates sketches of frogs is connected to another network which converts frog sketches to photographs. In this way, the model is a multi step process with two completely different operations working together to generate an image. One may draw a comparison to the brain, which contains thousands of operations, each with different functions, accumulating together to produce consciousness and feeling. The resulting frogs are an amalgamation of the model's constituent parts. Could you say these images are an abstraction of how the model is *thinking*? Is this model more or less intelligent because of the artist's distance from the output - and subsequent lack of control?

It employs a Pix2Pix network trained on translating contours to photos of frogs. A separate GAN is also trained on sketches of frogs. Both networks were trained, interacted, and exported with the software. The Pix2Pix dataset included 7,796 frog images (256x256)[41]. Whilst the dataset could have been compiled identically using the software's dataset builder (with image processing), this was not yet implemented when the work was created. As a result, the dataset was produced manually using a python-based OpenCV contour detection custom program. The GAN dataset featured 159,047 doodles of frogs (28x28) curated from Google's Quick Draw dataset[40].

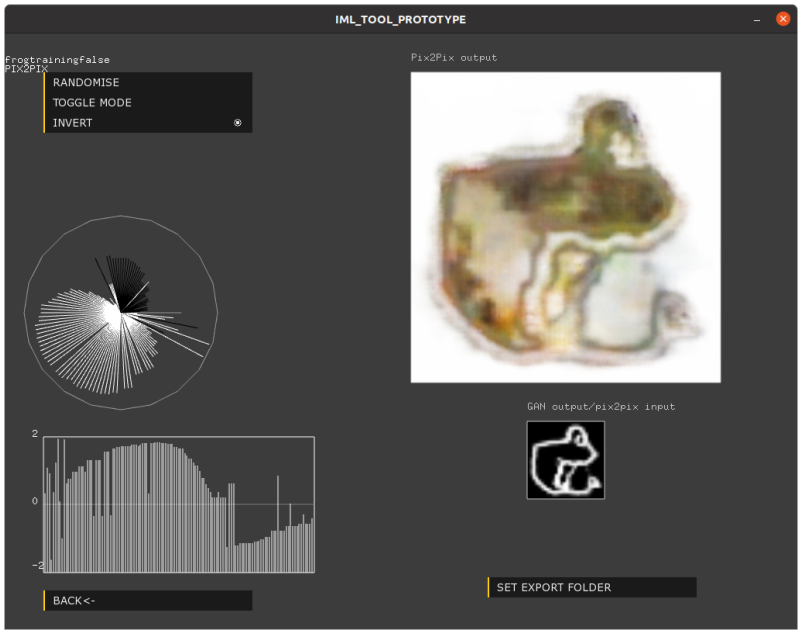


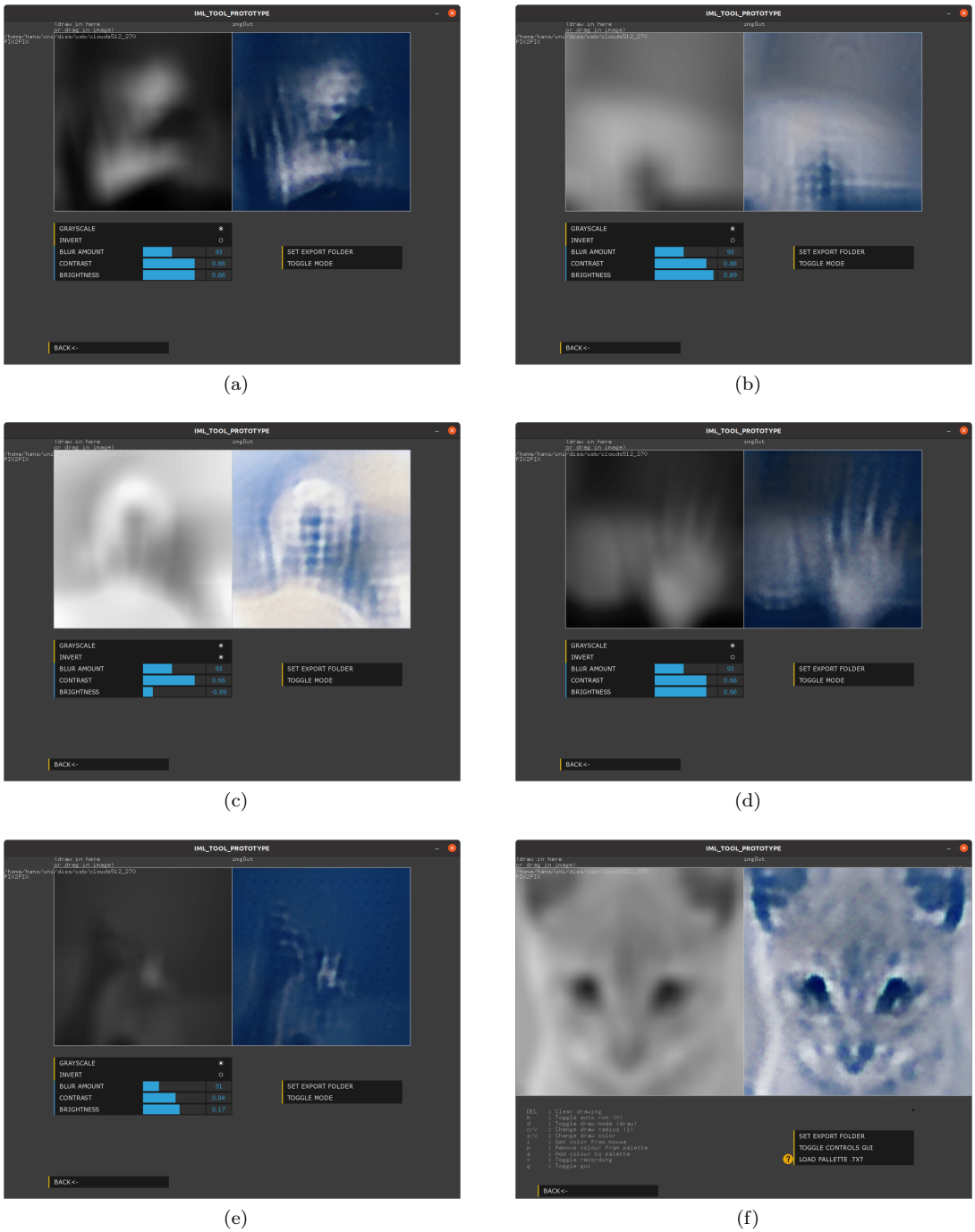
Figure 4.2: Neural Frogs Interaction

[1]The colours of the radial widget were slightly altered to gray (for negative values) and black (for positive values).

Figure 4.3: Neural Frogs.

Untitled - *Learning To See*[3] Inspired Work

Having drawn heavy inspiration from Memo Akten's Learning To See, I questioned the possibility of recreating a version of his work with the software. A Pix2Pix network was trained in the software on a cloud dataset[22] of 1,013 images, paired with the blurred and grayscaled counterpart. This augmentation was applied using the dataset building with image processing scene. The work constitutes an observer interacting with the webcam (within the software) to produce clouds. I was not happy with the results, however, as it seemed to perform better as a super-resolution solution than a cloud generator. In the future, this will be repeated on a more extensive dataset of clouds.



(a) (b) (c) (d) (e) (f)

Figure 4.4: Different parameter choices with the Learning To See adaption and the model in the draw scene (bottom right) to demonstrate its super-resolution capability.

Mode Collapse of a Moth

Mode Collapse of a Moth is a video of the training progression of a GAN generating moth images. I wanted to exploit the impact of a dynamic learning rate on a neural networks evolution. The aim was to produce training feedback that wasn't motionless or static. A variably winged insect was selected to be generated, with the intention of portraying movement through flight. In the video, changes in learning rate at certain intervals affect the pace in which the moths find their form. The moths are transformed from a soft noise into photo-realistic insects at a non linear rate. In doing so, the observer witnesses the life-cycle of the artificial moths; from a noisy and formless beginning, to a stable and coherent end. In between, the moths randomly fluctuate between turbulent and lucid states, much in the same way that nature does not present itself as a linear process. Mode collapse, where the generator produces the same output(s), is incurred before the network can finally reach stability - and the moths can reach their final form. Conventionally, mode collapse is a symptom of error. This work embraces mode collapse by defining it as the natural/evolutionary course of action for the network, and consequently, the moths.

Figure 4.5: Mode Collapse of a Moth (https://vimeo.com/647028909).

The GAN was trained on a dataset[10] of 2,120 moths re-scaled to 512x512. The training process

was performed on a separate machine (Windows) externally to the software - but using identical python files[2]. This was due to GPU limitations of my hardware. The *images* directory was then copied to a dummy model and the GIF was exported from within the software. The GIF was then trivially converted to video (also externally).

Lump - an Experimental Music Video

This works sets out to create an experimental music video for a snippet of a techno track (James Holden, Lump). A GAN was trained (on an arbitrary 64x64 dataset) for a very small number of epochs (<10) and exported in the software using the music sync functionality. This was repeated on the same network 9 times with different music sync parameters. A video editing software was then used to tile the videos.

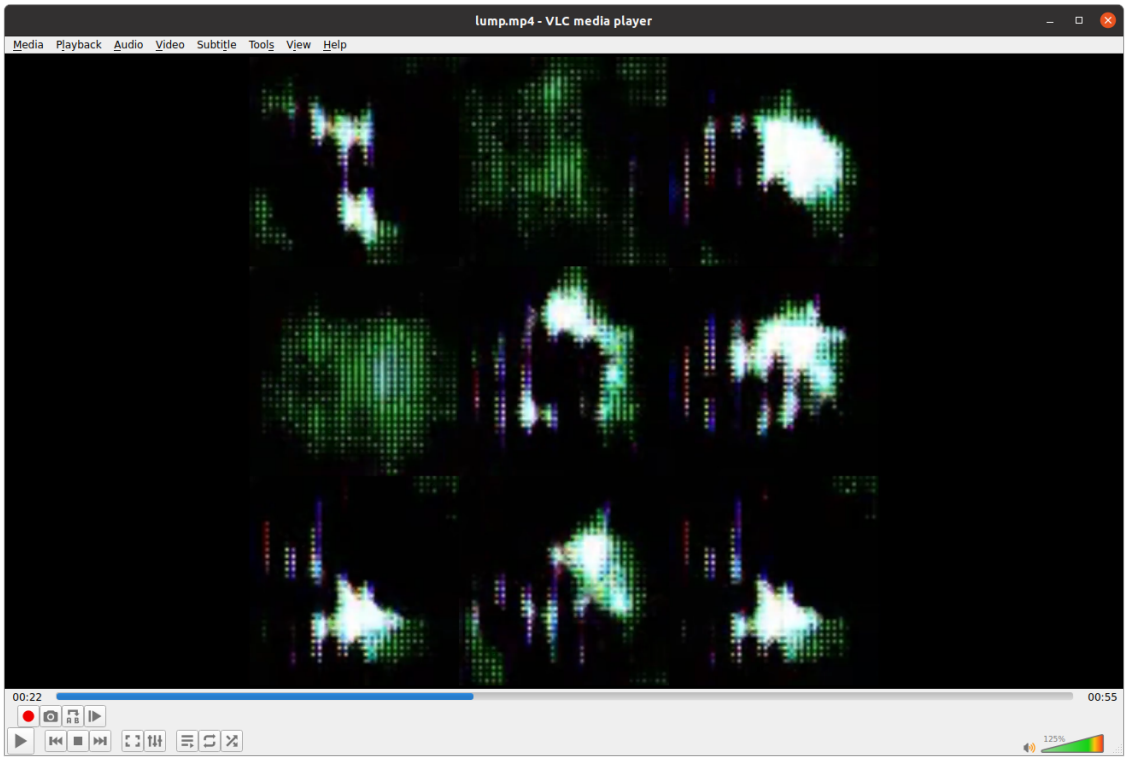
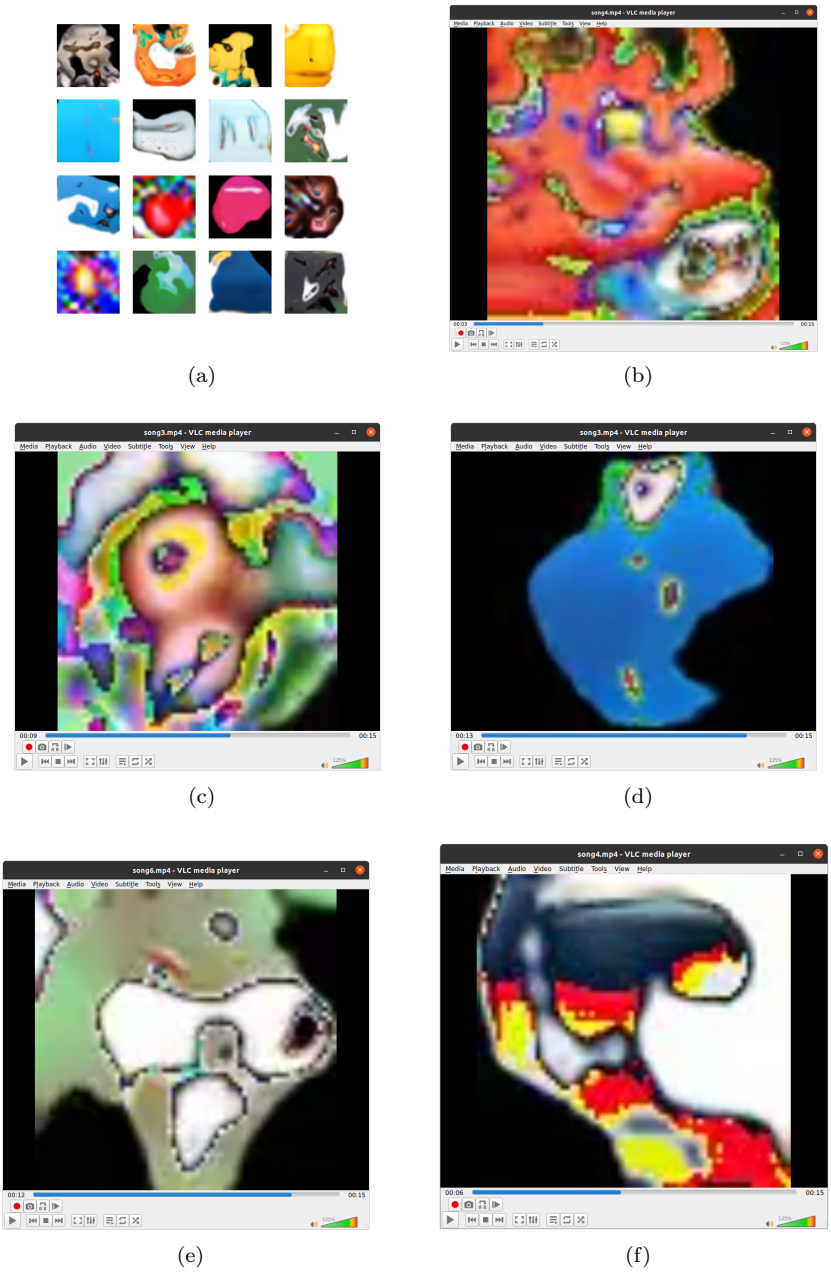


Figure 4.6: Lump. https://vimeo.com/647079347 - the video was poorly processed by Vimeo; the video on my local machine runs much more fluidly.

Emoji

Emoji is very similar to the lump music video: a network is trained in the software, and halted prematurely, on a dataset of emojis. The sync music functionality is then applied on an audio file generated from an online text to speech tool. The text repeatedly says "emoji". Due to the vibrant colours present in emojis, as well as facial elements, the resulting video has a very psychedelic feel to it. This work was intended as a bit of fun. A fully developed emoji GAN will likely feature in the play scene for the user to interact with (see Section 3.7).

[2]Except the `dcgan_builder.py` which included a 512x512 generator option featuring extra layers. This option will eventually be implemented verbatim into the main software, so this work (except GIF to .mp4 conversion, which could also be a capability in the future) will offer full capacity to replicate this work.

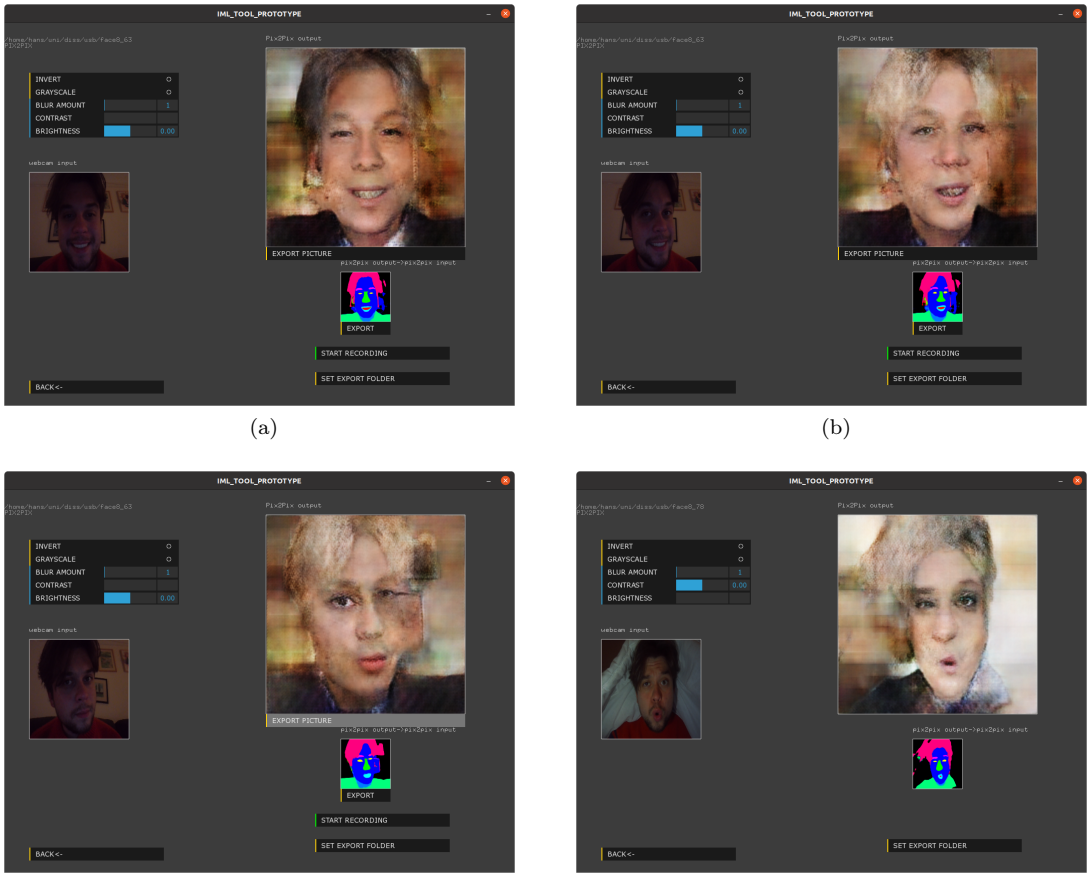


(a) (b) (c) (d) (e) (f)

Figure 4.7: Emoji. Image generated during training (top left) and frames from the video. https://vimeo.com/647097700.

Artificial Reflection

Artificial reflection is an ongoing interactive work that reflects an artificial face onto the observer in real time. The motivation behind this piece was to experiment with the software's Pix2Pix2Pix2Pix functionality. In this way, it utilises an image to image translational model as input to another image to image translational model. The face generation model was trained using the software (in a similar pseudo-external fashion to Moth Collapse) on 23,000 segmented face photos (512x512) from the Celeb-A dataset[53]. For the input model, the same dataset was used but the image pairs were flipped and resized to 256x256. The dataset was paired external to the software, since the segmented dataset did not present itself in an acceptable format. It comprised of 12 colour encoded mask files that had to be processed as well as the target image files. Features that were extracted included: the nose (green), eyes (yellow), and mouth (light blue).The main network of artificial reflection also facilitates drawing (currently close to) photo-realistic faces using this encoded representation.



(a) (b)

(c) (d)

Figure 4.8: Artificial reflection being interacted with. The webcam input was captured in poor quality lighting, hence an attempt for clearer output by putting a duvet on my head (bottom right). With an established setup, the artificial reflections will be much clearer as there will be less noise picked up in the background.

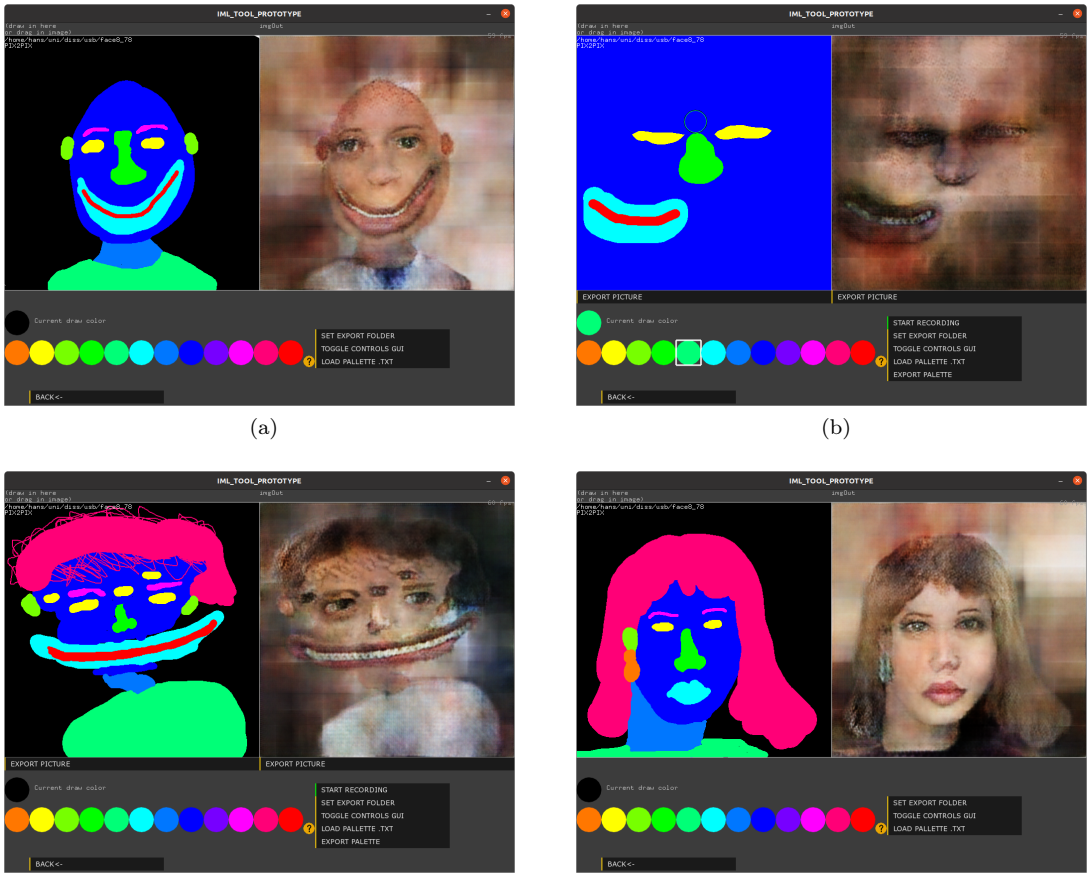
(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k) (l)

(m) (n) (o) (p) (q) (r)

(s) (t) (u) (v) (w) (x)

Figure 4.9: A sample of inputs (left), intermediary inputs (middle), and outputs (right) of Artificial Reflection.



(a) (b)

(c) (d)

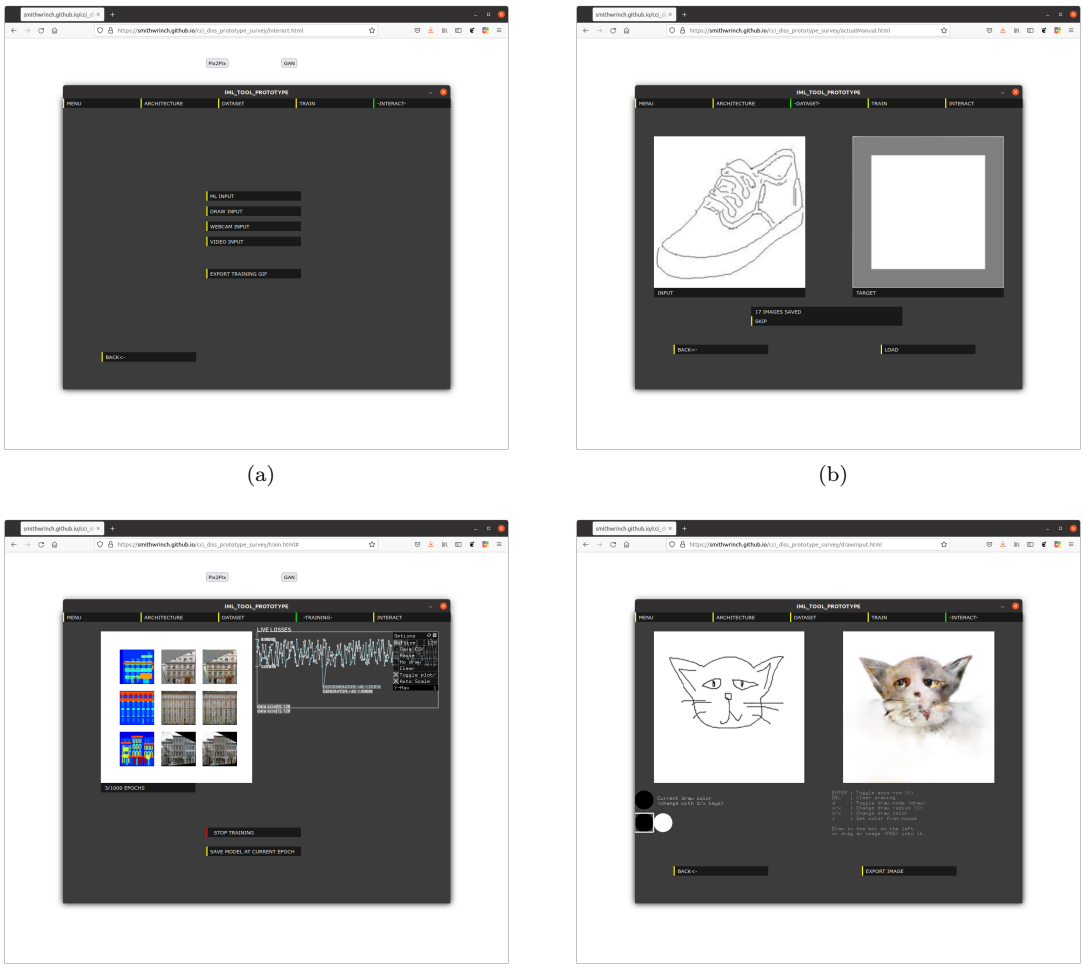
Figure 4.10: Face drawing interaction using the Artificial Reflection model. Due to bias[65] in the Celeb-HQ dataset, the work will not be officially released until this bias has been mitigated.

Chapter 5

Evaluation

5.1 Paper Prototype and Intermediary Survey

Part way through development, a survey was conducted where participants from ML and/or creative backgrounds were asked questions on a paper prototype of the software. The questions and results are outlined as follows:



(a) (b)

(c) (d)

Figure 5.1: Some scenes in the paper prototype. The prototype was a basic web page written in HTML, CSS, and Javascript. It had functionality to instruct the user using alert boxes as well as toggle between Pix2Pix and GAN variations of scenes (see top and bottom left). The full prototype is available at https://smithwrinch.github.io/cci_diss_prototype_survey.

Personal Information

In total, six people participated in the survey[1]. The participants were instructed to go through the prototype before beginning the survey. After the survey, they were invited to give extra feedback.

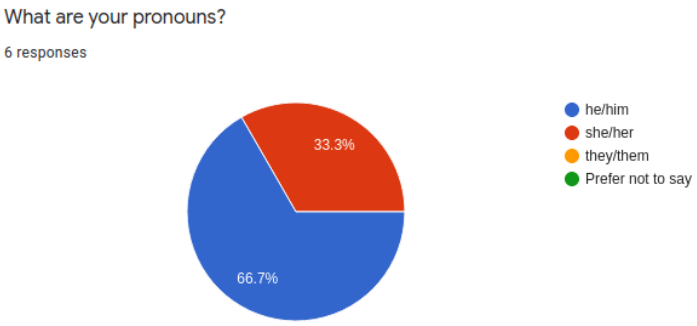


Figure 5.2: Question 1.

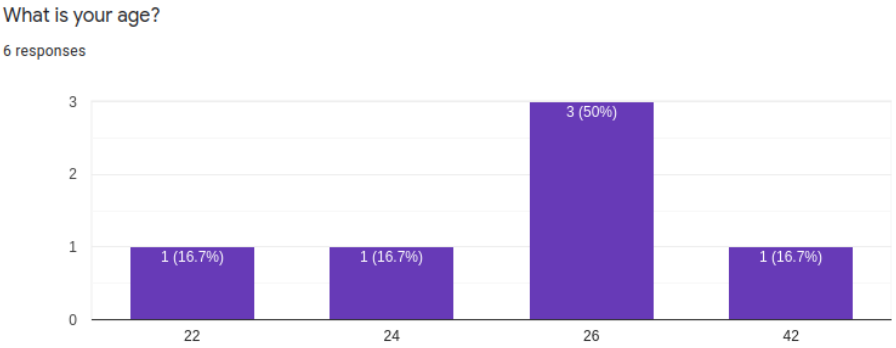


Figure 5.3: Question 2.

Figure 5.4: Question 3.

[1]DISCLAIMER: one participant's answers were removed from the study. This was due to supplying clearly erroneous and non-serious answers. They answered 1 for everything (including age) and in the general feedback section, wrote: "Wow thanks for this free text box thingy, I can post pictures of my pets or something".

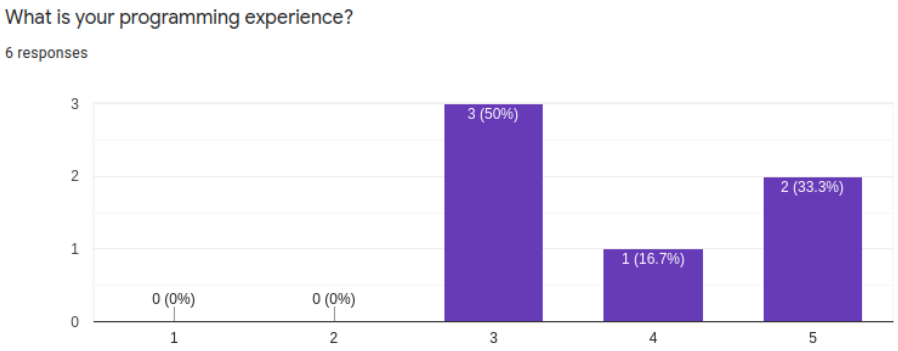


Figure 5.5: Question 4.

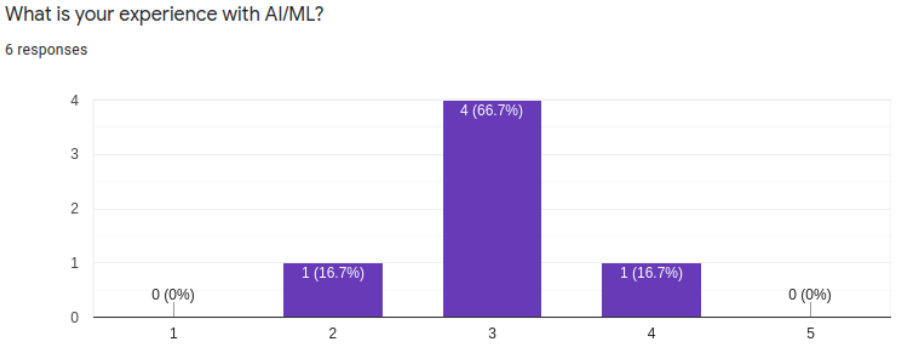


Figure 5.6: Question 5.

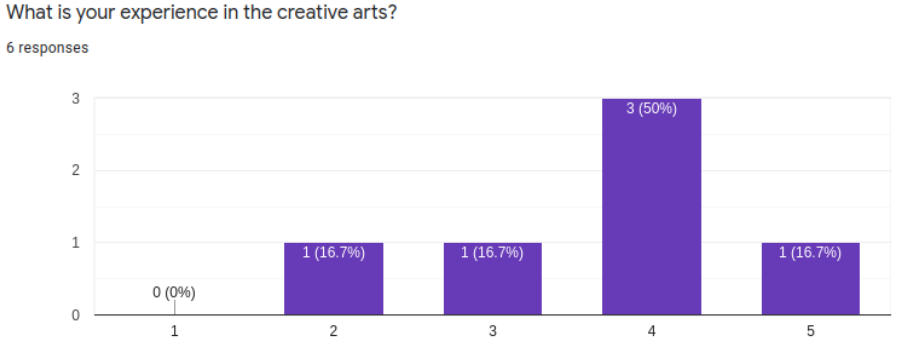


Figure 5.7: Question 6.

Main Survey

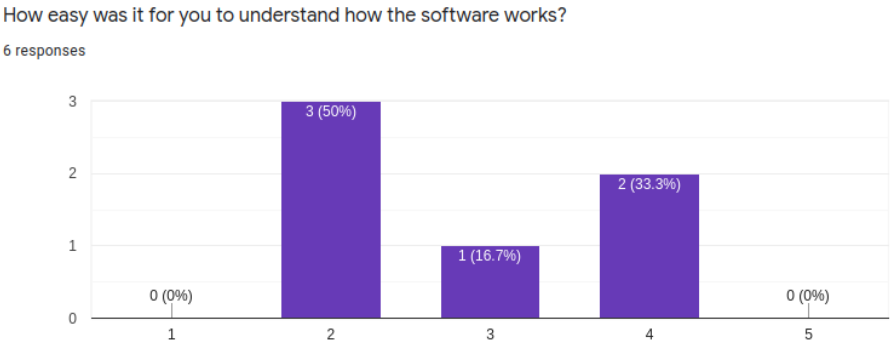


Figure 5.8: Question 7.

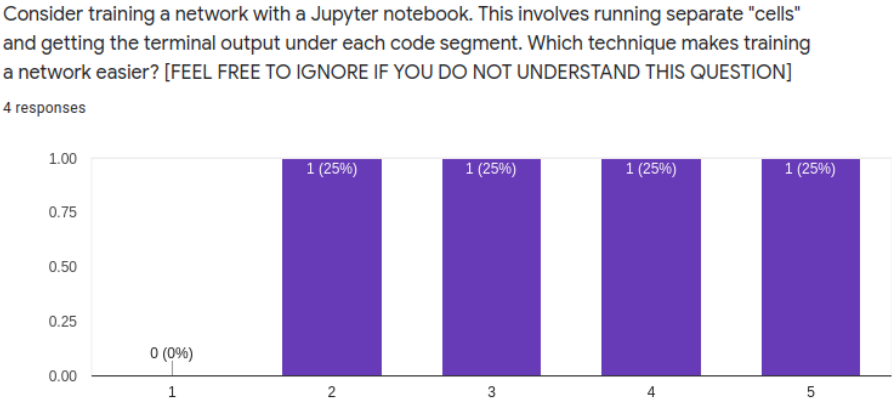


Figure 5.9: Question 8.

```
In  [42]: classifier.fit(X_train, y_train, batch_size=10, epochs=10)

          Epoch 1/10
          5686/5686 [==============================] - 1s 143us/step - loss: 0.6925 - acc: 0.5190
          Epoch 2/10
          5686/5686 [==============================] - 1s 134us/step - loss: 0.6925 - acc: 0.5190
          Epoch 3/10
          5686/5686 [==============================] - 1s 138us/step - loss: 0.6925 - acc: 0.5190
          Epoch 4/10
          5686/5686 [==============================] - 1s 135us/step - loss: 0.6925 - acc: 0.5190
          Epoch 5/10
          5686/5686 [==============================] - 1s 150us/step - loss: 0.6925 - acc: 0.5190
          Epoch 6/10
          5686/5686 [==============================] - 1s 139us/step - loss: 0.6925 - acc: 0.5190
          Epoch 7/10
          5686/5686 [==============================] - 1s 152us/step - loss: 0.6925 - acc: 0.5190
          Epoch 8/10
          5686/5686 [==============================] - 1s 143us/step - loss: 0.6925 - acc: 0.5190
          Epoch 9/10
          5686/5686 [==============================] - 1s 149us/step - loss: 0.6925 - acc: 0.5190
          Epoch 10/10
          5686/5686 [==============================] - 1s 146us/step - loss: 0.6925 - acc: 0.5190

Out[42]: <keras.callbacks.History at 0x2e62850e908>
```

Figure 5.10: Question 8 was accompanied with this picture of a Jupyter notebook's training procedure.

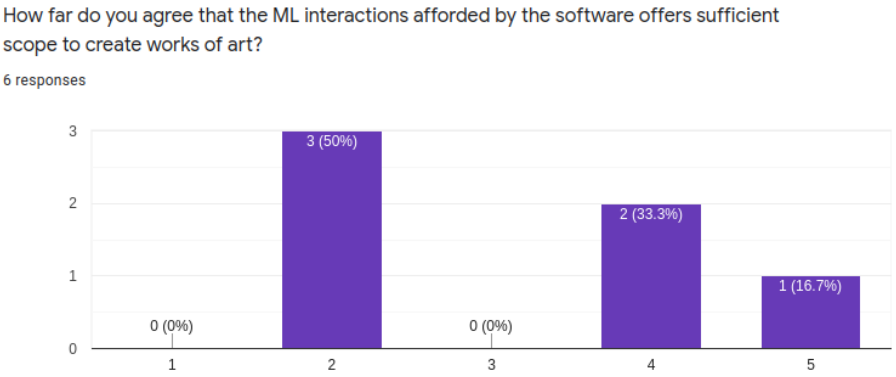


Figure 5.11: Question 9.

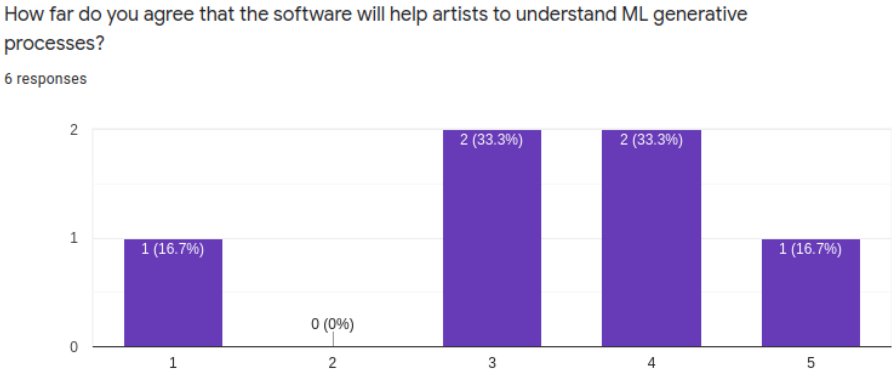


Figure 5.12: Question 10.

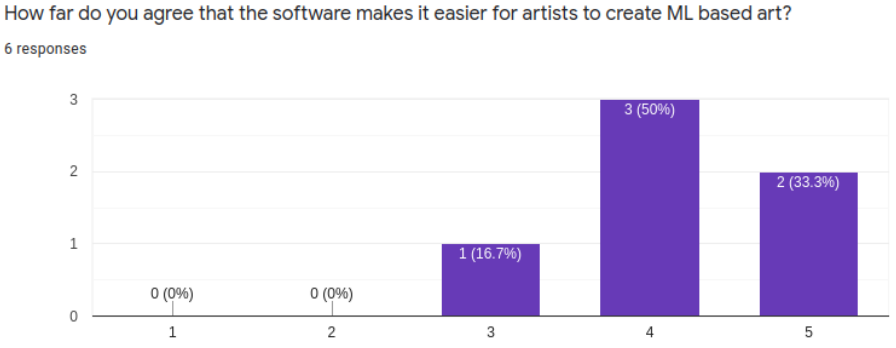


Figure 5.13: Question 11.

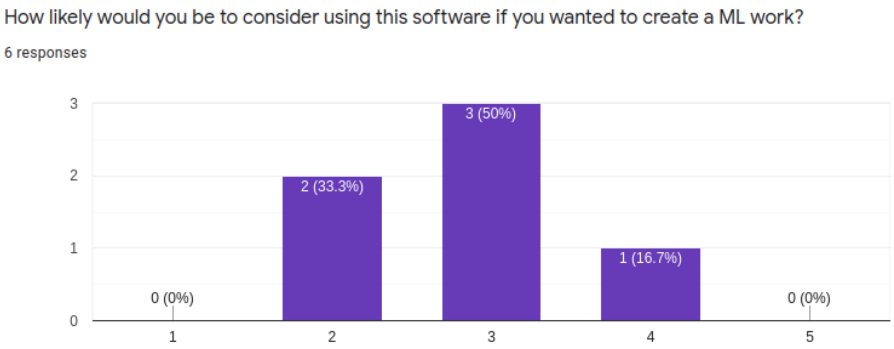


Figure 5.14: Question 12.

5.1.1 Summary

The participants displayed a middle to high level of programming experience, with an average value of 3.83/5. Among them, experience in the creative arts was greater than in AI/ML, with averages 3.67/5 and 3.00/5 respectively. In question 7, the majority of participants found it somewhat difficult to understand how the software worked. Comparing the software to a Jupyter notebook, 4 participants answered each option except from "Jupyter Notebooks make training much easier". This yielded an average of 3.50; slightly agreeing that the proposed software makes training easier. Half the participants somewhat disagreed that the software offered sufficient scope to create artworks, with the other two participants agreeing and the final participant strongly agreeing - giving an average of 3.17. There was a varied response to question 10. One participant strongly disagreed that the software will help artists understand ML generative process, and another strongly agreed. The rest were split between 3 (impartial) and 4 (agreeing). On average, participants slightly agreed with the statement, with an average value of 3.33. The last two questions had the least varied response of the main survey. In question 10, most of the participants agreed that the software makes it easier for artists to create ML based art with an average of 4.17 (slightly more than agreeing). The final question asked whether the participant would consider using the software to create an ML work. The answers were moderately skewed towards "not considering", culminating an average of 2.83.

5.1.2 Discussion

The intermediary survey clearly indicated that work needs to be applied on the paper prototyped version of the software. Firstly, although there was an average agreement that training a network was easier in the software than in a Jupyter notebook, the program aims to achieve a much easier training process for all participants - in order to accomplish the goal of increasing artists' understanding of ML. This work argues that if the software is only slightly easier than a conventional training procedure, which is convoluted and black-boxed, it is unlikely to democratise ML art production to a sufficient standard. Interestingly, exactly half of the participants independently suggested elemental information through a hover interaction. In response, help widgets were implemented throughout the program (see Figure 3.3). On hover, the widgets indicate tips about an element and its effect on the model. By incorporating these help widgets alongside the more technical interfaces, it also substantiates a reaction to questions 7 and 10. Further cause for concern came from the results of questions 9 and 12. With half of the participants disagreeing that there was sufficient scope for artists to create works of art, the software either lacked sufficient modes of output, or the modes of output were insufficiently communicated to the user. To combat the former, an extra mode of interaction was implemented for image to image translational models (see Section 3.6.2). This enabled users to use as input a separate Pix2Pix (or other I2I) model. At the time of writing, the input can be controlled through webcam interaction, with plans to support the other modes of Pix2Pix interaction. There is also an intention to increase the generative model collection, with text generation and style transfer functionality. Additionally, a revised version of the software introduced custom models (see Section 3.6.3). To address the latter, numerous works of art were created using the software to demonstrate its capacity for creative production (see Chapter

5.2). One could argue however, that the artworks might not generalise to other outputs; the works supply insufficient evidence of a wholly encompassing creative utility. That is, although the examples prove the existence of diverse modes of export, it may be possible that artists can only produce creative artefacts that are explicit variations of the supplied examples. Only another user study, in collaboration with artists, can prove this. Inadequate communication with the participants surrounding the software's creative potential may have also been a direct result of a flawed prototype. For example, some features were not implemented as illustratively as others. This included the video input (P2P) and music sync (GAN) interactions. Since these interactions present the most static interfaces, further dynamic elements (such as an example video output or additional HTML pages) might have been beneficial to illustrate their capability. Further to this, the play scene of the prototype was defined using an alert box. This was because it was not considered important, at the time, to the core software. Better communication could have been expressed if the play scene had a dedicated HTML file with a video detailing its interactions and export capability. The study also received several instances of feedback on the prototype itself such as concerns on being unable to interact with certain buttons. Indeed, one participant questioned why the training wasn't working[2]! This suggests that there may have been a problem with the format of the prototype. Seeing as it was a paper prototype (which expresses a rudimentary form), the study might have benefited from a greater abstraction of the software. Put another way, it is possible that the prototype was reviewed by participants as a close-to-established (potentially web-based) product - although this was clearly stated otherwise at the beginning of the survey. A more abstract prototype would have emphasised the user experience design (rather than UI design) and distracted the participant away from the prototype, to focus on the software it represents. An alternative prototyping mechanism A crucial piece of feedback may have outlined why the data was skewed against the participants consideration in using the tool for their own ML works (question 12), as well as previous question responses. It stated: "*Consider your target group and audience: will your users already have a previous experience with and understanding of machine learning processes and architectures? In the case of a ML layman, it might be useful to provide more information about the learning- and model options you're providing. I must admit, when I was prompted with the model options, it was a lack of feed-forward*[3] *that made me confused about what model to choose*". He addresses one of the greatest challenges in the software's design: catering to the target user (as an example, see Section 3.3). Since the scope of target users extended to technical and non-technical artists, each necessitating significantly different design requirements, the software is prone to suffer a pitfall by failing to cater to one (or either). Although having declared ML experience (a value of 3), he remained confused with aspects of the software's functionality, as well as stating that he was unlikely to use the software for his artworks (a value of 2). Additionally, the participant who was the most experienced with AI/ML was also unlikely to use the software for their work (she also gave a value of 2). Consequently, there is reasonable grounds to claim the prototype failed to cater to artists experienced in ML. Responding to the case of a "*ML layman*", the remaining four participants averaged a lower ML experience (2.75), with the average member of the group claiming they were slightly likely to use the software (3.25). A hypothesis was put forward that the software was more likely to be used by users with less experience in ML than those with more experience. To test this hypothesis, a Pearson's correlation coefficient was calculated. Comparing answers from question 7 and 12, the r value was -0.420. This indicates a weak correlation between inexperience in ML and the likelihood of a participant using the software in future work. Due to the small sample size, however, the hypothesis cannot be accepted or rejected until more data is acquired. Finally, whilst the survey on a paper prototype provided useful feedback in the development process, it cannot substitute a study conducted using the actual software. Once the installation protocol is configured, there is plans to reproduce this survey using a working implementation of the software.

[2]This inspires an implementation of a web-based version of the software, perhaps through tensorflow.js or similar.
[3]Feed-forward: being able to tell what action will be performed with the next interaction.

5.2 Conversation with a Technical Artist

To compliment the intermediary survey, further feedback was received from Mathew Plummer-Fernández, an artist and educator who utilises ML in his works. An informal interview was conducted on the latest version of the software, after suggestions from the survey were implemented. Before conversation began, Matthew was shown a video walkthrough of the software[4] and Chapter 5.2 of this document. This section shows a sample of our conversation.

Me: After watching the video, do you feel that you understand the software?

Matthew: Yes

Me: Do you think artists who didn't know what Pix2Pix or GAN meant would have a difficult time following the software?

Matthew: I think so. Just in my experience as an educator, I tend to introduce students to RunwayML which at least has a landing page for each model so that you have a description and an image what the model does, and that is the first step before jumping into the UI

Me: It's interesting you mentioned RunwayML as that is the main competitor for the software. Do you notice any differences between the two?

Matthew: I think what's nice is that your UI changes depending on the model and that you have all of your parameters accessible. Whereas RunwayML hides a lot of parameters and I think that's because it makes the UI a bit more uniform across models, but obviously every model has a different set of parameters and those parameters can be really useful for creatives to test out different ideas

Me: I found myself in a battle accommodating people experienced in ML and accommodating people who were inexperienced, like if I gave too many parameters it might frighten artists without technical experience

Matthew: The parameter conversation is interesting because parameters do offer a lot of creative freedom that, even for an experienced ML artist is hard to know what results you're going to get for different ranges of parameter [..] generally when I make colabs[7] for students to use, I put a comment next to each parameter to say [for example] 0.5 works well because I think sometimes as a beginner, if you don't know what to do at least you have a preset. So having tried and tested set of presets might be a way around the parameter problem

Me: There's a basic version of that implemented actually, where if a user selects a GAN for example, the default the parameters are set to what's on the example DCGAN file in Tensorflow, the one trained on MNIST. So if there was maybe a pointer for the first time [user] to train on MNIST and see how things go. There's also help widgets which I implemented after feedback just to explain what they [the parameters] mean. Do you reckon that also helps answer that problem?

Matthew: I think help buttons are really useful, sometimes you just need plain English explanation of what the parameter term means. So for example batch size, it sounds obvious but people will have different understandings of what batch size might mean

Me: How do you reckon the training process compares with RunwayML, is it different or the same?

Matthew: I really liked your training tools because RunwayML almost treats training as a separate activity and they've kind of created a different workspace and workflow. [Also] A lot of it is based around training StyleGAN and it already comes with a lot of presets using transfer learning. I think there approach is a little bit discouraging because it feels quite constricted

[4] https://vimeo.com/647610046 - password: "*IML*"

Me: Is there an argument that it makes it easier because it is constricted for artists without a technical background?

Matthew: The styleGAN model is an obvious choice, I think lots of artists can enjoy making a styleGAN out of a dataset that they've collected so that ones always going to be a popular tool to include. I really liked that yours, for instance, had a Pix2Pix model-

Me: -I was honestly schocked that Runway did not have Pix2Pix because in my head its the easiest form for an artist to control their expression

Matthew: Yeah I think that, to me, that was one of the highlights [you including] the Pix2Pix architecture, and then you had a tool for interacting with those models

Me: Eventually there's going to be a python module that follows the same arguments that the OF program sends out so users can plug in their own architectures

Matthew: Oh amazing [...]

Me: [...] Have you ever come across people inputting GANs into Pix2Pix or Pix2Pix into Pix2Pix before?

Matthew: Not really, I've just seen some early stuff like the Pikachu drawings so it was quite nice to see you expanding - almost like chaining up different Pix2Pix models

Me: Eventually I'd like to have a long daisy chain of Pix2Pix models and just seeing what comes out, I think it would be quite interesting

Matthew: I think the drawing tools are really interesting, I teach some computational art students that are currently at that interception of having been classically trained in drawing and painting and fine arts, now learning to use machine learning tools. I think Pix2Pix has a future you know, you can almost treat it like a Procreate app where you're drawing on an Ipad and it's computing a picture. I think, in a way, where the interface could be improved is thinking more about the drawing tool side because trying to select a colour from a limited palette and brush size is quite a cumbersome- [task]

Me: -In that scene you can use keyboard shortcuts to increase and decrease the brush size as well as load custom palettes from text files but I definitely agree with you. There needs to be more focus on drawing functionality just so we can kind of provoke that instance of classically trained artists and ML

Matthew: I do feel like digital drawing has advanced so much and so many illustrators now do their drawing digitally because there's auto-fill and stuff like that, you can draw the outline and fill in the colours from a dropdown palette or something like that, so I think that's where you could innovate on the UI. [...]

Me: [...] Do you think there is enough creative utility [in the software] to create works that are beyond the scope of the examples given [before the conversation started]? So like, if an artist picks up this tool, would they just create versions of works I've already made - or is there more creative potential?

Matthew: It's difficult to say if some of these models break down if you try more adventurous stuff. [..] I think definitely a more professional artist with more time on their hands would pursue all those potential hacks and tricks in different ways of using the tool. So instead of just going to a tried and tested Pix2Pix face drawing tool to draw face, they might try draw something totally different and see if some of those facial features are translated to that new thing. Like could you draw a tree on the face drawing one, to see if it produced a tree made out of like flesh with eyeballs for example. So I think artists will definitely start playing around testing the limitations of the software.

Me: It would be an interesting user study to create. Would you consider using the software yourself or recommending the software?

Matthew: Yes I'd definitely consider recommending this software, I think it would be great for my computational art students who have that background in drawing and illustration. This would be a natural way of integrating their skills.

Me: I almost think the software could be split into two parts which is the training, the dataset configuration and the architecture configuration - and then interaction. So, maybe there could be a collaboration of more technical artists and artists who are more classically trained - and they could use two separate parts of the software.

Matthew: Yeah absolutely, the thing is I think the training phase is still so weirdly appealing to the artist because they might have a particular aesthetic that they had in mind, and if they can train a Pix2Pix architecture on their particular aesthetic then they can almost make their own drawing tool. I guess that opens up another question about - there's like an initial phase that's often not discussed which is getting the data in the first place, that's like a skill in itself. Finding ways to generate Pix2Pix training data can often involve [garbled: other similar processes?..] But in a way, it's almost like for the artist, reverse engineering their process. Say if you draw buildings in a certain way, would you have to build first the dataset where you've taken existing photographs of buildings. And then manually outlined them yourself and then from then onwards, it would take a photo of a building and output it in their visual style. So I think the beginning phase and the end phase [of ML development] could both be satisfying for someone that comes from a more arts background.

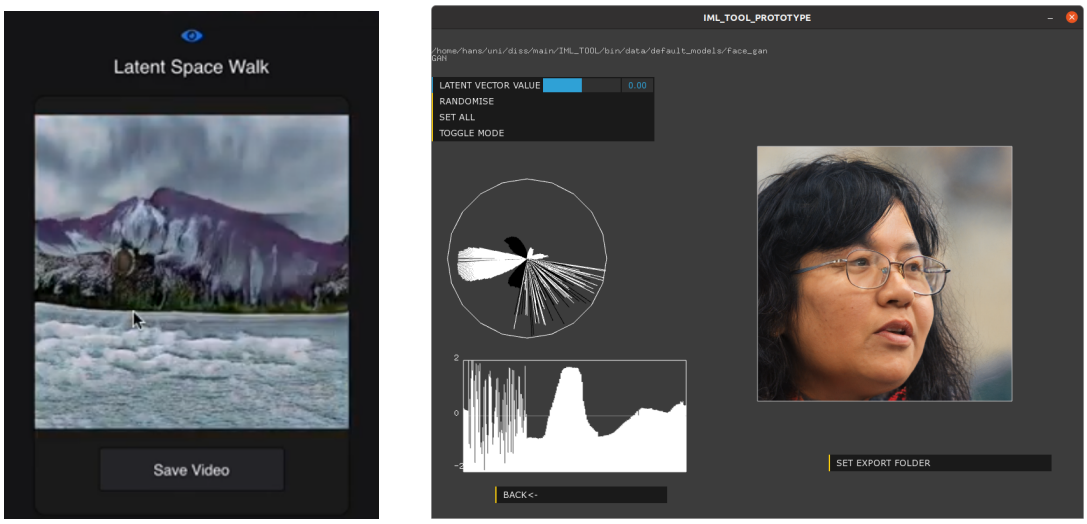
Brief Discussion

The conversation began with a suggestion of a UI with greater feedforward in the model selection scene. For inexperienced users, this would mean they would have access to a better understanding of the function of a GAN and Pix2Pix network. Considering this was also mentioned by a participant in the previous section, this will definitely be added in future work. Positive feedback was given about the help widgets, which served as a response to negative feedback in the intermediary survey. There was also positive feedback on the training workflow, specifically the IML approach (flexibility to move between states) compared to RunwayML. The conversation made clear that Pix2Pix was a positive addition to the software, stating that the drawing tools have lots of potential. It revealed insights into the creative potential of classically trained artists integrating their skills with ML tools. After discussing the various mechanisms employed by digital illustrators to facilitate their work, there leaves little doubt that pre-existing drawing interfaces could be implemented in the Pix2Pix draw scene. Consequently, this promotes further enrichment of the software's drawing capability. Furthermore, addressing previous concerns surrounding output generalisation, Matthew gave an example of an adaptation of the example artworks; suggesting there is a greater creative scope than the works illustrated in chapter . Finally, Matthew raised a very interesting point about an artist drawing satisfaction from both the developmental and inference phases. After some thinking, this provides a motivation to make the dataset and architecture configuration states more engaging. Further experiments on the user experience design of these scenes will therefore be performed.

5.3 RunwayML Comparison

It is important to compare the software to the most established end-to-end tool for artistic ML generation. The most obvious difference involves RunwayML's failure to innately support image to image translational models (both training and interactions). As aforementioned in the previous conversation, this foregoes a lot of creative potential. Additionally, the proposed software offers a significant increase in customisability and control. The software further mitigates any expense to the inexperienced artist through help widgets. An argument could be made, therefore, that the software promotes a greater understanding of ML as a process than RunwayML by abstracting less of the development procedure. A key difference between the two lies in the fact that RunwayML cannot be trained on a local machine. This forces users to pay per minute of training to access training functionality. As a result, this software provides a greater level of democratisation for ML and the arts, since users are capable of training locally for free. A counter claim could be made, however, stating that a user requires sufficient hardware to access the full software, especially with larger networks. Whilst in RunwayML, the training process is hardware independent. A connection to a Google Colab[7] instance could be instantiated within the software to address this claim. The software also benefits through its IML user experience design. Compared to RunwayML, which consists of a one directional workflow, the proposed software allows the user to

reach and manipulate different states without losing training progress. This substantiates richer interactions with the model and more control on the intended output. Lastly, the software provides a more extensive toolkit of interactions with RunwayML, with an example illustrated in Figure 5.15.



(a) (b)

Figure 5.15: The latent space walk interaction for a GAN in RunwayML (left) and the explore latent space scene (right). Other than generating pseudo-random outputs, the latent space walk is the only interaction, for a GAN, afforded to the user in RunwayML. In contrast, the proposed software supports functionality to sync a GAN to music, perturb subject to motion through a webcam, and manipulate the latent vector directly via a richer set of tools. Comparing the latter to the latent space walk, the software provides a clearly upgraded alternative.

5.4 Personal Evaluation

Overall, I am very pleased with the current state of the software. I proved to myself that the software significantly enhanced my workflow in generating ML art and I found a lot of fun playing with model interactions; often I would get too engrossed and become distracted from development. Hopefully, both statements are also proven to the reader, after considering the seven works that were created in conjunction (and simultaneously) to the software. There remains areas for improvement, however. Considering the seven works, there were numerous instances where processing had to be outsourced - such as with the Artificial Reflections dataset. After a richer engagement in the dataset (and architecture) configuration was concluded to be beneficial, from the conversation with an artist, improvements could be made with a more extensive dataset building implementation. Of course, the base Pix2Pix dataset building scene is yet to be completed! Another improvement to the dataset scene would involve explicit dataset information. I often found myself forgetting what dataset I was currently loaded, or how many images it contained. This could be resolved by having a supplementary "dataset viewing" state, where the user could toggle through images in the loaded dataset and be informed of its metadata. Alternatively (or additionally), the entire model state could be displayed to the user using a side panel - an interface that was suggested by a participant in the survey. The side panel could include details on the architecture, training status, and dataset - and could be appended to as the user progresses through stages. It could resolve concerns I have surrounding users forgetting the current architecture configuration. Things brings me to the most crucial point: it is impossible to know how any design suggestions will impact user experience and software efficacy until further case studies have been performed. In my opinion, the dominant flaw in this work comes at the expense of not implementing a functional installation protocol in time. Without a capacity for installation, the software cannot be sufficiently evaluated; at least one study on artists using the software as a whole (not as a paper prototype or viewing a video) is necessary. Furthermore, a lack of installation immediately fails to achieve the first two objectives set out in Section 1.1. However, a case could be made these objectives have been met, in theory,

having been outlined in Section 3.8. That is, the objectives would have been met, given more time. Personally, I choose refute this case until installation has been implemented in practice. I would also argue that the rest of the objectives have been met except six, which remains undetermined until further studies are performed.

Chapter 6

Conclusion

In conclusion, this work has proposed a functional end-to-end IML tool for artists that provides dataset and architecture configuration capabilities, training and live training feedback, and various modes of interaction for two kinds of generative artificial neural networks - with a focus on customisability and usability for artists from all technical backgrounds. The software was designed with a multi-faceted approach to allow user freedom over end-to-end or reduced engagement, through external model incorporation. An argument has also been made stating that the model is an improvement on the, at the time of writing, most popular ML generative tool currently used by artists: RunwayML. Two case studies saw varied responses from artists, promoting a need for further study. Integrating ML methods into an artist's classical skillset remains a significant challenge both for the user experience and software design. There is hope that this work is a step in the right direction in producing a fully democratised ML environment for artists.

6.1 Future Work

Since the proposed software is currently an early stage prototype, the future work comprises an extensive list.

1. Installation capability
2. Tertiary case study
3. Develop python module to allow community to create custom models efficiently
4. Implement Load Architecture scene
5. Implement Pix2Pix dataset builder scene
6. Increase dataset and architecture configuration engagement
7. Increase configuration options, such as automatically save epoch every x epochs
8. Implement a logging system so a user can track parameter changes and current configuration
9. Full screen mode for interactions, or customisable exhibition scene
10. GAN Motion scene improvements, such as using kernels to represent latent vector elements
11. Increase Pix2Pix input to all other interactions (not just webcam) for Pix2Pix
12. Allow daisy chaining for ML models (instead of just 2 connected)
13. Include different learning rates for discriminator and generator
14. Include different input and output dimensions for Pix2Pix
15. Include other generative models such as variational autoencoders, style transfer, and text generation

Bibliography

[1] URL: https://affinelayer.com/pixsrv/.

[2] Anna ridler uses ai to turn 10,000 tulips into a video controlled by bitcoin. URL: https://www.itsnicethat.com/articles/anna-ridler-digital-art-280619.

[3] Memo Akten, Rebecca Fiebrink, and Mick Grierson. Learning to see: you are what you see. In *ACM SIGGRAPH 2019 Art Gallery*, pages 1–6. 2019.

[4] Mikael Alafriz. Introducing "lucid sonic dreams": Sync gan art to music with a few lines of python code!, 2021. URL: https://towardsdatascience.com/introducing-lucid-sonic-dreams-sync-gan-art-to-music-with-a-few-lines-of-python-code-b04f887

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[6] Francisco Bernardo, Mick Grierson, and Rebecca Fiebrink. User-centred design actions for lightweight evaluation of an interactive machine learning toolkit. *Journal of Science and Technology of the Arts*, 10(2):2–25, 2018.

[7] Ekaba Bisong. *Google Colaboratory*, pages 59–64. Apress, Berkeley, CA, 2019. doi:10.1007/978-1-4842-4470-8_7.

[8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[9] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobb's journal of software tools*, 3, 2000.

[10] Gunnar Brehm, Patrick Strutzenberger, and Konrad Fiedler. Phylogenetic diversity of geometrid moths decreases with elevation in the tropical andes. *Ecography*, 36(11):1247–1253, 2013.

[11] Meredith Broussard, Nicholas Diakopoulos, Andrea L Guzman, Rediet Abebe, Michel Dupagne, and Ching-Hua Chuan. Artificial intelligence and journalism. *Journalism & mass communication quarterly*, 96(3):673–695, 2019.

[12] Cathrine Kieu Trang Bui. Exploring bias against women in artificial intelligence: Practitioners' views on systems of discrimination. Master's thesis, 2021.

[13] Joy Buolamwini. Aspire mirror, 2015. URL: http://www.aspiremirror.com/.

[14] Joy Buolamwini. Upbeat walls, 2016. URL: https://www.youtube.com/watch?v=Pqc9tWQdW8U.

[15] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018.

[16] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[17] Pablo Samuel Castro. Ganterpretations. *arXiv preprint arXiv:2011.05158*, 2020.

[18] Stephen Cave, Kate Coughlan, and Kanta Dihal. " scary robots" examining public responses to ai. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 331–337, 2019.

[19] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

[20] Kate Crawford. Artificial intelligence's white guy problem. *The New York Times*, 25(06), 2016.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[22] Soumyabrata Dev, Yee Hui Lee, and Stefan Winkler. Color-based segmentation of sky/cloud images from ground-based cameras. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(1):231–242, 2016.

[23] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[24] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[25] John J Dudley and Per Ola Kristensson. A review of user interface design for interactive machine learning. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 8(2):1–37, 2018.

[26] Rebecca Fiebrink. Wekinator 2.0. 2015.

[27] Rebecca Fiebrink. Machine learning education for artists, musicians, and other creative practitioners. *ACM Transactions on Computing Education (TOCE)*, 19(4):1–32, 2019.

[28] Adrian Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *International Computer Music Conference (ICMC)*, 1997.

[29] Swetava Ganguli, Pedro Garzon, and Noa Glaser. Geogan: A conditional gan with reconstruction and style loss to generate standard layer of maps from satellite images. *arXiv preprint arXiv:1902.05611*, 2019.

[30] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[31] Philippe Gellman. Blockchain: The new art house. *ITNOW*, 63(3):18–19, 2021.

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[34] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In *International conference on machine learning*, pages 2672–2680. PMLR, 2019.

[35] Aaron Hertzmann. Can computers create art? In *Arts*, volume 7, page 18. Multidisciplinary Digital Publishing Institute, 2018.

[36] Clarice Hilton, Carlos Gonzalez Diaz, Ruth Gibson, Phoenix Perry, Rebecca Fiebrink, Michael Zbyszynski, Nicola Plant, Bruno Martelli, Marco Gillies, et al. Interactml: Node based tool to empower artists and dancers in using interactive machine learning for designing movement interaction. 2020.

[37] MORI Ipsos. Public views of machine learning. *Report title*, 92, 2017.

[38] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[39] Sergio Izquierdo. Cppflow. URL: https://github.com/serizba/cppflow.

[40] Jonas Jongejan, Henry Rowley, Takashi Kawashima, Jongmin Kim, and Nick Fox-Gieg. The quick, draw!-ai experiment. *Mount View, CA, accessed Feb*, 17(2018):4, 2016.

[41] @jonshamir. Frog dataset, 2018. URL: https://github.com/jonshamir/frog-dataset.

[42] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[43] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[44] Kimon Kieslich, Birte Keller, and Christopher Starke. Ai-ethics by design. evaluating public perception on the importance of ethical design principles of ai. *arXiv preprint arXiv:2106.00326*, 2021.

[45] Eugenia S Kim, Christian Sandor, Jayson Haebich, and Alvaro Cassinelli. Playing with soma: Speculating on the physical body and somatic practice of ai. In *Art Machines 2: International Symposium on Machine Learning and Art 2021 (AM2)*, pages 31–38. School of Creative Media, City University of Hong Kong, 2021.

[46] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pages 1857–1865. PMLR, 2017.

[47] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[50] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[51] Kevin J Liang, Chunyuan Li, Guoyin Wang, and Lawrence Carin. Generative adversarial network training is a continual learning problem. *arXiv preprint arXiv:1811.11083*, 2018.

[52] Qingyun Liu, Huihuang Zhao, Ying Wang, Feng Zhang, and Manimaran Ramasamy. Sketch to portrait generation with generative adversarial networks and edge constraint. *Computers & Electrical Engineering*, 95:107338, 2021.

[53] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018.

[54] Alexander Mathiasen and Frederik Hvilshøj. Fast fréchet inception distance. *arXiv e-prints*, pages arXiv–2009, 2020.

[55] Louis McCallum and Mick S Grierson. Supporting interactive machine learning approaches to building musical instruments in the browser. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 271–272, 2020.

[56] James E McDonough. Singleton design pattern. In *Object-Oriented Design with ABAP*, pages 137–145. Springer, 2017.

[57] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[58] Arthur I Miller. 12 jun-yan zhu's cyclegan turns horses into zebras. 2019.

[59] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[60] Donald A. Norman. *The Design of Everyday Things*. Basic Books, Inc., USA, 2002.

[61] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. URL: https://developer.nvidia.com/cuda-toolkit.

[62] Leila Ouchchy, Allen Coin, and Veljko Dubljević. Ai in the headlines: the portrayal of the ethical issues of artificial intelligence in the media. *AI & SOCIETY*, 35(4):927–936, 2020.

[63] Zhaoqing Pan, Weijie Yu, Bosi Wang, Haoran Xie, Victor S Sheng, Jianjun Lei, and Sam Kwong. Loss functions of generative adversarial networks (gans): opportunities and challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(4):500–522, 2020.

[64] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Gaugan: semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!*, pages 1–1. 2019.

[65] Vinay Uday Prabhu, Dian Ang Yap, Alexander Wang, and John Whaley. Covering up bias in celeba-like datasets with markov blankets: A post-hoc cure for attribute prior avoidance. *arXiv preprint arXiv:1907.12917*, 2019.

[66] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[67] Anna Ridler. Myriad (tulips), 2018. URL: http://annaridler.com/myriad-tulips.

[68] Anna Ridler. bloemenveiling, 2019. URL: http://annaridler.com/bloemenveiling.

[69] Anna Ridler. Mosaic virus, 2019. URL: http://annaridler.com/mosaic-virus.

[70] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[71] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[72] Anurag Sarkar and Seth Cooper. Towards game design via creative machine learning (gdcml). In *2020 IEEE Conference on Games (CoG)*, pages 744–751. IEEE, 2020.

[73] Julia Schnitzer. Generative design for creators–the impact of data driven visualization and processing in the field of creative business. *Electronic Imaging*, 2021(3):22–1, 2021.

[74] Mark Segal and Kurt Akeley. The opengl graphics system: A specification (version 1.1), 1999.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] Irwin Sobel. History and definition of the sobel operator. *Retrieved from the World Wide Web*, 1505, 2014.

[77] Jason Tsay, Todd Mummert, Norman Bobroff, Alan Braz, Peter Westerink, and Martin Hirzel. Runway: machine learning model experiment management tool. In *Conference on Systems and Machine Learning (SysML)*, 2018.

[78] David Uthus, Maria Voitovich, and RJ Mical. Augmenting poetry composition with verse by verse. *arXiv preprint arXiv:2103.17205*, 2021.

[79] Qiu Yu, Jamal Malaeb, and Wenjun Ma. Architectural facade recognition and generation through generative adversarial networks. In *2020 International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, pages 310–316. IEEE, 2020.

[80] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.